EIT DIKTATENSERIE 5

# informatiestructuren, bestandsorganisatie en bestandsontwerp

ACADEMIC SERVICE

Z 550

# INFORMATIESTRUCTUREN, BESTANDSORGANISATIE EN BESTANDSONTWERP



Het Economisch Instituut Tilburg is een wetenschappelijke instelling, verbonden aan de Katholieke Hogeschool te Tilburg. Naast het verrichten van onderzoek op het terrein van de economische wetenschappen verzorgt het EIT cursussen op het gebied van de informatica en de statistiek.

Medewerking aan deze publikatie werd verleend door de docenten van het EIT basisjaar.
Deze herziene versie werd gerealiseerd door:

J. J. van Griethuyzen
P. Langendoen
J. M. H. Nieuwhof
G. F. Wilmink

EIT DIKTATENSERIE 5

# informatiestructuren, bestandsorganisatie en bestandsontwerp

ACADEMIC SERVICE

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Informatiestructuren

Informatiestructuren, bestandsorganisatie en bestandsontwerp / [gerealiseerd door J.J. van Griethuyzen ... et al.]. -
Den Haag : Academic Service. - Ill. - (EIT diktatenserie ; 5)
Met lit. opg.
ISBN 90-6233-019-3
SISO 365.2 SVS 8.12.3 UDC 681.3.016 UGI 200
Trefw.: bestandsorganisatie

1e druk 1975
2e herziene uitgave 1977
3e ongewijzigde druk 1978
4e ongewijzigde druk 1980
5e ongewijzigde druk 1982
6e ongewijzigde druk 1983
7e ongewijzigde druk 1985

Uitgegeven door: Academic Service
Postbus 96996
2509 JJ Den Haag
Druk: Krips Repro Meppel
Bindwerk: Meeuwis, Amsterdam
Omslagontwerp: JAM Gauw
ISBN 90 6233 019 3

INHOUDSOPGAVE

# 1 INLEIDING

Een informatiesysteem zullen we omschrijven als het geheel van activiteiten, hulpmiddelen en methoden waarmee een organisatie aan zijn informatiebehoefte tracht te voldoen.
In informatiesystemen moeten grote hoeveelheden gegevens opgeslagen, verwerkt en opgezocht worden. Deze gegevens beschrijven objecten (begrip, betekenisvolle eenheid), (personen, goederen, projecten e.d.), die in een organisatie aanwezig zijn. Van deze objecten worden alleen de kenmerken beschreven die voor het functioneren van de organisatie van belang zijn. Bijvoorbeeld: object: persoon, kenmerken: naam, adres, leeftijd.
In deze syllabus zullen we zien hoe deze opslag van gegevens kan geschieden, daarbij zowel het gewenste gebruik van de gegevens (toegankelijkheid) als de huidige opslagmogelijkheden beschouwend. Tevens zullen we enige aandacht schenken aan de samenhang, structuur van gegevensverzamelingen.
Belangrijk is het dan een onderscheid te maken tussen de informatiestructuur, de logische structuur die de samenhang van de gegevensverzameling beschrijft en de fysieke of opslagstructuur, waarin vastgelegd is hoe een bepaalde logische structuur afgebeeld is op beschikbare geheugensystemen.
De logische structuur is een gegeven dat vanuit de probleemstelling en probleemanalyse naar voren komt (personen werken in een afdeling, een artikel is een onderdeel van een samengesteld artikel etc).
De fysieke structuur wordt zodanig ontworpen of - als standaard software beschikbaar is - gekozen dat de logische structuur efficiënt op de beschikbare achtergrondgeheugens afgebeeld wordt. De overwegingen die bij dit proces een rol spelen zijn voornamelijk: zuinig geheugengebruik en snelle toegankelijkheid.
Bij het programmeren van informatieverwerkende processen poogt men thans steeds meer de fysieke structuur niet in het programma door te laten dringen. De logische structuur is een onderdeel van de probleemoplossing en is dan ook verweven in het programma. Doel van dit streven is de programmeur te ontlasten van bestandsorganisatie-trucs en de programma's onafhankelijk te maken van de fysieke eigenschappen van dit ene computersysteem.

Bij programmeertalen als Cobol en Fortran treft men nog wel elementen van de fysieke structuur in de programmatekst aan, alhoewel men er naar streeft de gekozen opslagstructuur zoveel mogelijk vast te leggen in de 'karweibesturingstaal'.
Bij data base management systemen (Mark IV, Infol, TDMS) zijn opslagstructuur en logische structuur wel volledig gescheiden.

In deze syllabus zullen achtereenvolgens aan de orde komen: basisbegrippen, enkele logische en fysieke structuren, opslagmedia en veel gebruikte opslagstructuren, bestandsontwerp en data base management systemen.

# 2 ENKELE BASISBEGRIPPEN

## 2.1 *Karakters*

De kleinste logische bouwstenen voor informatievastlegging zullen voor ons karakters zijn, dat wil zeggen de letters van een alfabet, de cijfersymbolen 0-9 en een hier niet nader bepaald aantal 'speciale' karakters als leestekens, rekenkundige symbolen enz. De afbeelding van deze bouwstenen op de kleinste fysieke bouwstenen, te weten bits zal niet aan de orde komen, behoudens de vaststelling dat voor de afbeelding van ieder karakter een vast aantal bits (meestal 6 of 8 bits, samen byte te noemen) nodig is.

## 2.2 *Items*

Met deze bouwstenen kunnen we de voor een probleem relevante, logische informatie-eenheden of items opbouwen en wel de naam van een item en de waarde van een item ('value', gegevenswaarde). Afhankelijk van de aard van het item kan de waarde numeriek (een getal) zijn, of alfabetisch (een naam of woonplaats) of alfanumeriek (een codewoord samengesteld uit letters en cijfers). De naam van een item (attribuut, gegevensklasse, 'property') wordt veelal niet expliciet vastgelegd en, indien dit wel het geval is, meestal met letters alleen. Het vastleggen van itemwaarden alleen (een 'homogene' informatieverzameling) impliceert een stilzwijgende afspraak over de volgorde van items en vereist de aanwezigheid van een speciale 'nulwaarde' (blank) om aan te kunnen geven dat een bepaald gegeven niet bekend is. Voor de fysieke vastlegging van een item in een veld (rubriek, 'field') kan enige conversie van de externe representatie plaatsvinden; zo worden decimale getallen veelal binair in de machine vastgelegd, de nog gebruikelijke datumschrijfwijze wordt geconverteerd naar de voor vergelijking handiger (en sinds kort genormaliseerde) representatie jaar-maand-dag, een volledige tekst wordt misschien gecomprimeerd door invoering van afkortingen of weglating van klinkers enz. Deze conversieproblemen zullen hier echter niet besproken worden. De lengte van een item is het aantal karakters, waaruit zijn fysieke vastlegging bestaat.

## 2.3 *Records*

De een bepaald verband hebbende items worden als regel verzameld tot een zogenaamd recordtype, bijvoorbeeld omdat het gegevens zijn die betrekking hebben op eenzelfde persoon of eenzelfde apparaat of eenzelfde complex van activiteiten ('entity' of 'object'). Is de naam van de entity expliciet in het record opgenomen, dan heet dit centrale item meestal sleutelitem of recordsleutel; de waarde van het sleutelitem heet de s l e u t e l ('key') omdat dit gegeven als het ware de toegang geeft tot andere bij de entity horende gegevens. Soms zijn meer sleutels nodig om een record éénduidig te bepalen (vergelijk het doel van naam, voorletters, geboortedatum).
Voorbeeld: Bij een personeelsafdeling worden over personeelsleden ('entities' of 'objects') de volgende gegevens (attributes, properties, eigenschappen die het object kenmerken) bewaard:

entity : personeelslid
properties: naam, adres, afdeling, aantal kinderen
values : jansen, dam 18 amsterdam, magazijn, 3.

Hierbij zij opgemerkt dat de entity of het recordtype 'personeelslid' is en dat de verzameling values ook wel aangeduid wordt als het record of het logical record. Afhankelijk van het resultaat van de bestudering van een probleem kan men besluiten tot het ene uiterste, namelijk om alle items behorende bij een sleutelitem in één record te plaatsen (dat dan wel erg groot kan worden) of tot het andere uiterste, namelijk een groot aantal records slechts bestaande uit het sleutelitem en telkens één ander item, of tot een oplossing tussen deze twee uitersten. Een bepaald gegeven bij een zeker item zal dan meestal via de sleutel in één van de records gezocht moeten worden. Het is echter ook mogelijk dat het gevonden moet worden via een verwijzing (wijzer, 'pointer, reference') in één van deze records (vergelijk de situatie in onze telefoongidsen, waarin we Bouw- en Woningtoezicht vinden via Gemeentelijke Instellingen).

De fysieke vastlegging van logical records gebeurt door 'b l o k k e n' van een aantal logical records tot een blok (physical record). Dit in verband met de efficiëntie van het gebruik van secundaire geheugensystemen, zoals we verderop zullen zien. Blokken worden met één invoer- of uitvoeropdracht gelezen òf geschreven (dit is: tussen secundair en primair geheugen gecopieerd); logical records moeten dan verder uit een blok geïsoleerd worden voor gegevensverwerking. In deze zin zijn kaarten uit een kaartsysteem te vergelijken met physical records, daar ze met één beweging gelicht worden.

De vastlegging van items in een logical record kan nog op verschillende manieren gebeuren:

1. de 'fixed' methode (of vaste veldlengte):
   - itemwaarden (eventueel spaties, indien niet gedefinieerd of bekend) worden in een vaste volgorde achter elkaar geplaatst,
   - er is een vast aantal items; iedere itemwaarde heeft zijn eigen, constante lengte (eventueel via opvullen met spaties),
   - het begin van iedere itemwaarde is zodoende een vast aantal posities na het begin van een logical record.

Dit is wel de meest gebruikte methode door zijn overzichtelijkheid en eenvoudige hanteerbaarheid. De recordlengte is eveneens vast en uiteraard gelijk aan de som van de itemlengten. Indien echter bij de verzameling soortgelijke records, de overeenkomstige items sterk in lengte variëren, dan geeft deze methode aanleiding tot inefficiënt geheugengebruik.
Voorbeeld: Indien de gegevens van het hierboven genoemde voorbeeld volgens deze methode vastliggen, zouden de records van twee personeelsleden er als volgt uit kunnen zien:

```
| jansen **** | dam * 18 * amsterdam * | *3 |

| denderen ** | wal ** 6 * ege ******* | 12 |

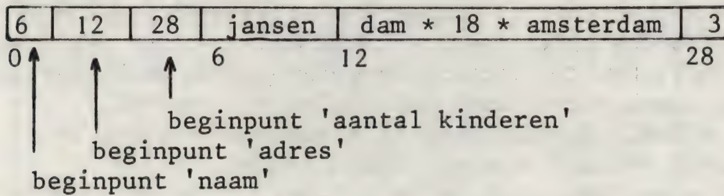
naam: max. 10 kar.  adres: max. 17 kar.  aantal
                                         kinderen:
                                         max. 2 kar.

*: stelt een spatie voor.
```

2. de 'indexed' methode:
   - itemwaarden worden in een vaste volgorde achter elkaar geplaatst,
   - er is een vast aantal items, maar itemwaarden hebben een variabele lengte,
   - daarom is ook de recordlengte variabel en moeten begin- en eindposities van iedere itemwaarde t.o.v. het recordbegin (dus per record) apart worden bijgehouden,
   - voor dit laatste worden dan ook hulpvelden (wijzers, pointers) in het record opgenomen die dienst doen om genoemde begin- en eindposities aan te geven. De wijzers worden als regel in het begin van een logical record geplaatst.

Deze methode is geschikt voor problemen waarbij veel gegevens van wisselende lengte voorkomen, zoals het geval is bij namen en adressen. (De rekenautomaat moet 'indirect addressing' of 'indexing' faciliteiten bezitten voor een efficiënte programmering van deze methode.)

Voorbeeld:




3. de 'separator' methode:
   - itemwaarden worden in een vaste volgorde achter elkaar geplaatst, hebben een variabele lengte,
   - zij worden van elkaar gescheiden door separators.

Deze methode heeft het bezwaar dat het opzoeken van een bepaald gegeven het karakter voor karakter afzoeken van het record vereist, hetgeen een tamelijk complex programma vereist indien er geen speciale hardware instructies zijn.

Voorbeeld:

| | | | | |
|---|---|---|---|---|
| jansen | ; | dam * 18 * amsterdam | ; | 3 |

Als separator is hier de ; gekozen.

4. de 'label' methode:
   - paren itemnaam-itemwaarde worden, eventueel in een willekeurige volgorde, achter elkaar geplaatst, itemnaam (label) en itemwaarde hebben een variabele veldlengte,
   - deze paren worden van elkaar gescheiden door separators.

Deze methode is een uitbreiding van de vorige in die zin, dat noch het aantal itemwaarden constant hoeft te zijn, noch de volgorde. Het opzoeken van een bepaald gegeven is echter nog eens zo ingewikkeld.

Voorbeeld:

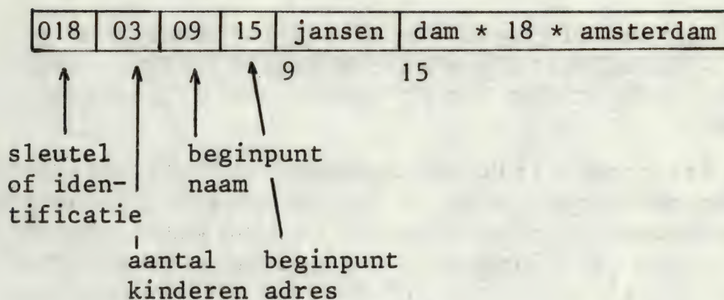
| | | | | |
|---|---|---|---|---|
| adres, dam * 18 * amsterdam | ; | naam, jansen | ; | aantal kinderen, 3 |

Als separator tussen paren is weer ; gekozen.
Als separator binnen paren is , gekozen.

De twee laatste methoden worden niet veel gebruikt. Mengvormen van deze vier methoden, vooral van de eerste twee, komen ook voor.

Voorbeeld: mengvormen van 1 en 2:

Stel elk personeelslid heeft ook nog een 3-cijferig identificatienummer. Dan zou men volgende recordopbouw kunnen kiezen:




Opmerkingen:
1. Variabele veldlengte geeft veelal aanleiding tot een variabele lengte van het logische record. Dit hoeft echter niet.
Evenzo geeft opblokken van logische records met variabele lengte tot een fysisch record (blok) veelal aanleiding tot blokken met variabele lengte. Ook dit is niet noodzakelijk.
2. Bij de eerste drie methoden bepaalt de volgorde de betekenis van de waarde; de naam van het item wordt immers niet opgeslagen. Deze vaste volgorde zal ook in de programma's die met deze records werken (vragen, muteren) een rol spelen. Men kan dit voorkomen door bij de verzameling records als extra informatie de betekenis van de volgorde op te nemen. Het enige verschil met methode 4 is dan dat de namen van de items slechts eenmaal opgenomen hoeven te worden en niet per record, omdat de volgorde van items in alle logische records dezelfde is.

## 2.4 *Bestanden*

Een geordende verzameling logical records, die dezelfde structuur (aard en vastlegging van items) bezitten, heet een b e s t a n d (file). De ordening kan het gevolg zijn van de wijze waarop het bestand oorspronkelijk opgebouwd wordt, maar ook van het sorteren van logical records op grond van de daarin voorkomende sleutel(s). Een bestandsgrootte, dat wil zeggen het aantal logical records, kan als regel slechts bij benadering opgegeven worden.
De fysieke vastlegging geschiedt meestal op een bepaald type informatiedrager (v o l u m e), bijvoorbeeld een magneetband (tape-reel) of schijfeenheid (diskpack). Wanneer op zo'n informatiedrager meerdere bestanden worden vastgelegd (uiteraard duidelijk van elkaar gescheiden), dan spreekt men van een multifile volume; vereist omgekeerd een bestand meerdere volumes, dan spreekt men van een multivolume file. Aan een informatiedrager wordt met behulp van een etiket veelal een externe bestandsnaam toegekend voor de menselijke identificatie.

Een verzameling bestanden wordt wel databank (data base) genoemd, al wordt deze term ook wel gereserveerd voor databanken, die op één volume opgeslagen zijn en waarin ieder item slechts één keer voorkomt.

Een bestandsorganisatie wordt gekenmerkt door de ordening van logical records in een bestand en door de afbeelding van de logical records op de fysieke informatiedragers. De keus van een bepaalde bestandsorganisatie wordt bepaald door specifieke probleemeisen ten aanzien van toegankelijkheid voor en kosten van het aanbrengen van wijzigingen en aanvullingen, voor het terugvinden van informatie, door beveiligingsaspecten en door de beschikbare apparatuur.

Bestanden worden onderworpen aan bewerkingen, die als volgt te rubriceren zijn:
- toevoegen (inserting) van records aan een bestand (het opbouwen van een nieuw bestand moet als bijzonder geval hiervan gezien worden),
- veranderen (updating) van (bepaalde delen van) sommige records in een bestand,
- verwijderen (deleting) van bepaalde records in een bestand,
- opzoeken van gegevens in een bestand op grond van hetzij kennis van de sleutel (we zullen dit eenvoudige proces 'searching' noemen) of op grond van de waarden van één of meer items ('retrieval' te noemen; searching is dan retrieval op grond van het sleutelitem),
- het sorteren van de records van een bestand.

De eerste drie bewerkingen veranderen iets aan het bestand; de vierde bewerking niet. Deze laatste is echter de basisoperatie: men moet eerst het record 'vinden' alvorens te kunnen veranderen of verwijderen.
In samenhang met deze bestandsbewerkingen worden de volgende begrippen gebruikt:
- bestandsaktiviteit (file activity). Hiermee wordt bedoeld het percentage logical records in een bestand dat tijdens het gebruik van een bestand betrokken is bij één of meer van deze bewerkingen
- bestandsverandering (file volatility) ook wel veranderingsgraad genoemd. Hiermee wordt bedoeld het percentage logical records in een bestand dat tijdens het gebruik betrokken is bij updating alleen
- bestandsverloop (file turnover) of vervangingsgraad. Hiermee wordt bedoeld het percentage logical records dat per periode wordt verwijderd en/of vervangen
- bestandsgroei (file growth). Hiermee wordt bedoeld de procentuele verandering in het totaal aantal logical records van een bestand per periode (dag, maand enz.).

Voorbeeld: als 10% van alle records uit een bestand wordt verwijderd en eenzelfde hoeveelheid nieuwe records wordt in dezelfde periode toegevoegd dan is de file turnover 10%.
In dat geval is de file growth nul en de file activity 20%.
Indien echter 30% van de records worden toegevoegd dan is de file turnover eveneens 10%, en de file growth 20% en de file activity 40%.
Worden tenslotte 20% van de records verwijderd en 10% toegevoegd dan is de file growth 10%, de file turnover 10% en de file activity 30%.

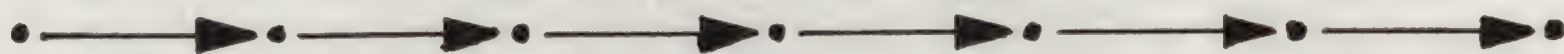
De voor ieder van deze bewerkingen benodigde tijd wordt bepaald door de grootte van het bestand en door de bestandsorganisatie.
Wat de 'beste' bestandsorganisatie is, hangt dan ook af van het soort bewerkingen waaraan een bestand het meest onderworpen wordt en van veiligheidseisen die gesteld kunnen worden.

# 3 LOGISCHE STRUCTUREN, OPSLAG-STRUCTUREN EN ACCESMOGELIJKHEDEN

## 3.1 *Logische structuren*

De logische structuur hangt samen met de wijze waarop men met de gegevens wil werken.
In principe is er maar één type logische structuur: de netwerkstructuur (graph, networkstructure). Vaak worden er echter een drietal genoemd: sequentiële structuur, boomstructuur (treestructure) en netwerkstructuur.

1. Sequentiële structuur:
Hierin wordt naar ieder element - behalve het eerste - verwezen door slechts één element, terwijl vanuit een element ook slechts naar één element verwezen kan worden.



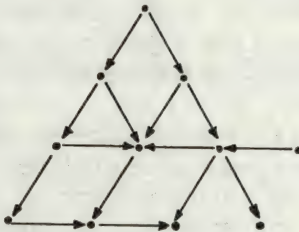
• : element, ⟶ : verwijzing

2. Boomstructuur:
Hierin wordt naar ieder element - behalve het eerste - verwezen door slechts één element, terwijl vanuit een element naar meerdere elementen verwezen kan worden.



Opm.: Ga na dat de sequentiële structuur een bijzonder geval is van de boomstructuur.

3. Netwerkstructuur:
Hierin kan naar ieder element verwezen worden door meerdere elementen, terwijl vanuit een element naar meerdere elementen verwezen kan worden.



Opm.: Ga na dat de boomstructuur een speciaal geval van de netwerkstructuur is.

## 3.2 *Opslagstructuren (fysieke structuren)*

De fysieke structuur of opslagstructuur van een bestand wordt in eerste instantie bepaald door de eigenschappen van de informatiedrager en daarnaast door de gewenste logische structuur.
In dit verband is het verwarrend dat dezelfde terminologieën voor fysieke structuren gehanteerd worden als voor logische structuren. Men dient onderstaande structureringsmogelijkheden dan ook op te vatten als methoden om logische structuren te implementeren.
In een ongestructureerd ('serial') bestand is er geen enkele relatie tussen de records van een bestand; zij staan bijvoorbeeld in de volgorde waarin de records toevallig aan het bestand toegevoegd zijn. Als regel zijn bestanden echter gestructureerd; men kan onderscheiden:

- Sequentiële structuren, wanneer de fysieke volgorde der records op grond van een of ander kenmerk tot stand is gekomen, als regel op grond van de waarde van de sleutel (oplopend of soms afnemend) en soms op grond van de activiteit (bewerkingsfrequentie) van de records. Er is geen relatie tussen recordsleutel en het (absolute) adres van het record op een informatiedrager. (Deze organisatie zal men veelal bij magneetbanden als informatiedragers toegepast zien.)
- Willekeurig toegankelijke ('direct') structuren, wanneer records zodanig op een informatiedrager geplaatst worden dat er wel een verband is tussen recordsleutel en (absoluut) adres van het record op een informatiedrager. De informatiedrager dient dan uiteraard adresseerbaar te zijn (bijvoorbeeld schijvengeheugens).
- Lijststructuren (sliertstructuur, 'list structure'), wanneer

records die logisch bij elkaar horen (bijvoorbeeld bij dezelfde sleutel) met elkaar verbonden zijn met behulp van wijzers ('pointers'), die in tegenstelling tot de vorige twee structuren een uitbreiding van ieder record met die wijzer vereisen. Bijzondere gevallen van lijststructuren zijn:

- enkelvoudige kettingstructuren (ketting in één richting), wanneer in ieder (behalve het laatste) record een wijzer opgenomen is naar het volgende op de een of andere wijze logisch samenhangende record;
- dubbele kettingstructuren (ketting in twee richtingen), wanneer in ieder record (behalve laatste en eerste) een wijzer staat naar het volgende en een wijzer naar het vorige record;
- ringstructuren (circular list), wanneer het laatste record in een kettingstructuur een wijzer naar het eerste bevat en het eerste (eventueel) een wijzer naar het laatste record (het 'eerste' record moet dan door een bijzonder kenmerk als zodanig herkenbaar zijn);
- boomstructuren, wanneer naar ieder record, behalve het eerste, door slechts één ander record verwezen wordt;
- netwerkstructuren, wanneer naar ieder record door één of meer andere records verwezen wordt.

### 3.3 *Accessmogelijkheden*

Om de verwarring te vergroten worden voor de accessmethoden (toegang tot het bestand) weer dezelfde termen gebruikt als voor de opslagstructuren (sequentieel, direct).
Daarom moeten we ook onderscheid maken tussen de structuur van een bestand, dat wil zeggen de opbouw en plaats van records in een bestand, en de toegang tot een bestand, dat wil zeggen de wijze waarop we aan een bepaald record komen.
Ten aanzien van de toegang tot een bestand is men tot op zekere hoogte afhankelijk van de structuur van het bestand (en zoals we later zullen zien van de aard van de informatiedrager). Bij ongestructureerde bestanden zit er voor het zoeken van een record niet veel anders op dan alle records achter elkaar te bekijken totdat men de gezochte records gevonden heeft (dit proces heet 'lineair' zoeken, linear search of sequentiële zoek). Als er meerdere records met dit zoekkenmerk zijn, dan moet men het hele bestand doorzoeken (full search).
Bij sequentieel gestructureerde bestanden kan men lineair zoeken tot de gewenste records gevonden zijn of de plaats waar ze hadden moeten staan gepasseerd is, hetgeen door vergelijking van sleutels te bewerkstelligen is (kan dit bij ongestructureerde bestanden ook?). In principe zijn ook andere methoden mogelijk (of de informatiedrager daarvoor geschikt is, zullen we later bekijken),

bijvoorbeeld 'binair' zoeken, waarbij we ieder interval waarin de gezochte records kunnen liggen (oorspronkelijk het hele bestand) steeds verder in twee stukken (niet noodzakelijk even groot) verdelen totdat de gezochte records al dan niet gevonden zijn. Nog een andere methode heet skip-sequential zoeken (getrapt sequentieel zoeken), hierbij gaat men - steeds sleutels vergelijkend - eerst met grote 'sprongen' door het bestand; wordt een passeren van de mogelijke plaats van de gezochte records geconstateerd dan gaat men met wat kleinere sprongen vanaf het beginpunt van de laatste sprong zoeken, enzovoort.
Bij direct gestructureerde bestanden heeft men per definitie toegang tot gezochte records zonder andere records te inspecteren. Dit kan gebeuren met een 'directe' zoek wanneer het adres van de gezochte records direct volgt uit de sleutel of met een 'indirecte' zoek, wanneer uit de sleutel via een of andere algoritme of via een tabel eerst nog het adres berekend moet worden. We komen hier later nog op terug.
Bij lijst-gestructureerde bestanden zal men bij het zoeken van een record voor het volgen van de wijzers weer 'voorgaande' records moeten inspecteren (tenzij de relatie tussen sleutel en recordadres bovendien nog via een aparte indexlijst op te sporen is. Ook hier is dan - mits er een volgorde is - binair zoeken etc. mogelijk.) Boomstructuren dienen zo te worden opgezet dat een zo klein mogelijke inspectiereeks nodig is.

Opmerking:
Van de vier hoofdbewerkingen op bestanden (opzoeken, wijzigen, weglaten, invoegen (zie hoofdstuk 2)), is het opzoeken in het voorgaande uitgebreid behandeld. Bij alle opslagstructuren, behalve de direct toegankelijke, is het opzoeken de basisbewerking voor de overige drie genoemde bewerkingen; bij de direct toegankelijke is het in elk geval de basisbewerking voor 'wijzigen' en 'weglaten'.
Indien men dan ook de mogelijke opslagstructuren vergelijkt met betrekking tot deze hoofdbewerkingen, dan betekenen tijdrovende opzoekmogelijkheden ook tijdrovende overige bewerkingen.

Daarnaast kan men het volgende opmerken over de overige bewerkingen:
- Wijzigen zonder dat de recordlengte wijzigt.
  Dit kan bij alle opslagstructuren even snel; bepalend is hier de zoektijd voor het record.
- Wijzigen met verandering van recordlengte.
  Bij opslagstructuren waarin de records fysiek naast elkaar liggen betekent een kortere lengte het ontstaan van gaten, die op de een of andere wijze geadministreerd moeten worden (vrije ruimte) of door opschuiven van de staart opgevuld moeten worden. Een vergroting van de recordlengte vereist zonder meer opschuiven.

Bij de overige opslagstructuren kan dit probleem zich ook voordoen, afhankelijk van de wijze van implementeren. In het volgende hoofdstuk wordt hierop nader ingegaan.
- Weglaten.
  Hiervoor gelden dezelfde opmerkingen als bij het wijzigen met verkleining van recordlengte.
- Invoegen.
  Dit kan eenvoudig geschieden bij een ongestructureerd bestand. Hier is het in feite geen 'invoegen', maar 'achteraan toevoegen'.
  Bij sequentiële structuren dient zich het opschuifprobleem weer aan.
  Bij direct toegankelijke structuren is invoegen geen probleem als elk record een eigen fysisch adres heeft.
  Bij lijststructuren komt invoegen in principe neer op het wijzigen van wijzers.

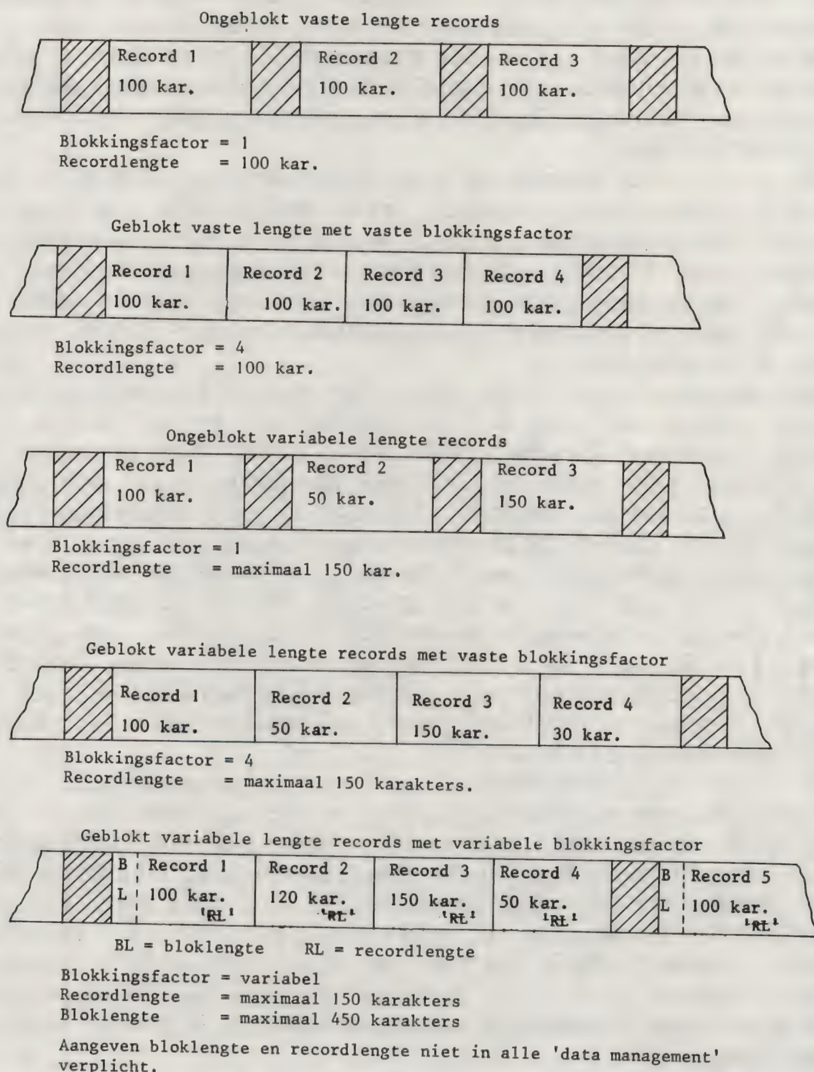
Resumerend kunnen we stellen dat het opschuifprobleem en het gatenprobleem zich altijd voordoen bij die opslagstructuren waar de structuur vastgelegd is in die fysische ordening van records (ongestructureerd en sequentieel gestructureerd), terwijl bij de overige structuren het probleem zich niet of in veel mindere mate voordoet.
De prijs die men daarvoor betaalt is een adresberekeningsalgoritme of administratie van wijzers.

Na de bespreking van informatiedragers zullen we aan de hand van enkele in de praktijk veel gebruikte bestandsorganisaties zien, dat bepaalde combinaties van structuren nuttig zijn.

# 4 INFORMATIEDRAGERS

De meest gebruikte informatiedragers in een computersysteem zijn ponskaart, ponsband, magnetische band, schijf en trommel.
Ponskaart en ponsband worden momenteel bijna uitsluitend gebruikt als invoermedium in tegenstelling met eerste en tweede generatie computers waar nog uitgebreide permanente ponskaartenbestanden gebruikt werden.
Ook als invoermedium is er een tendens om de ponskaart en ponsband te vervangen door snellere invoermedia zoals magnetische kaarten en magneetband, waarbij een directe registratie plaatsvindt via bijvoorbeeld een toetsenbord op deze magnetische geheugenvormen. Voor de opslag van permanente bestanden zijn de meest gebruikte informatiedragers de magnetische band, - schijf, - trommel en - stripgeheugens.
Deze secundaire geheugens hebben een gemeenschappelijke eigenschap: informatie wordt hierop opgeslagen in series van een groot aantal karakters. Een dergelijke serie wordt blok genoemd. Tussen de blokken is om technische redenen steeds een vrije ruimte noodzakelijk. Deze ruimte wordt blokhiaat of gap (blockspace) genoemd. Kiest men de bloklengte klein dan blijft een relatief groot deel van de beschikbare geheugenruimte onbenut in de vorm van de blokhiaten. Om deze reden zal men in vele gevallen bij records met niet te grote recordlengte er een aantal samenpakken tot een blok. Men spreekt dan van geblokt met als blokkingsfactor het aantal records per blok. Hier doet zich dan het geval voor van records die tegen elkaar geplaatst staan. Ingeval één record overeenkomt met één blok spreekt men wel van ongeblokt.
Een record is dus een eenheid vanuit gebruikersstandpunt, een blok daarentegen een eenheid vanuit technisch computerstandpunt. Deze begrippen worden nog wel eens door elkaar gehaald, hetgeen in de hand gewerkt wordt door de Engelse benamingen. Daarbij wordt het hier gedefinieerde record aangeduid als logical record, terwijl een blok dan genoemd wordt physical record. Via vertaling doen in het Nederlands dan weer de benamingen logisch record en fysisch record hun intrede. Eenvoudiger is echter de korte en duidelijk gescheiden benamingen record en blok aan te houden zoals hiervoor omschreven.
Op overeenkomstige manier wordt de bloklengte weer onderscheiden

in vaste bloklengte en variabele bloklengte. Soms geeft men hier nog een derde vorm aan als ongedefinieerde bloklengte (figuur 1). Wanneer men dit doet brengt men eigenlijk een splitsing in het begrip variabele bloklengte. Onder variabele bloklengte verstaat men dan een blok van variabele lengte echter voorafgegaan door een lengte aanduiding. Bij een ongedefinieerde bloklengte ontbreekt deze lengte aanduiding, zodat daar de lengte van een blok alleen bepaald wordt door de scheiding van het volgende blok in de vorm van het blokhiaat.




Figuur 1: Blokformaten.

Magneetbanden

De oudste vorm van secundair geheugen is het magneetbandgeheugen. Magneetbanden kunnen eenvoudig gebruikt worden voor blokken met variabele lengte aangezien directe adressering ontbreekt. De blokken zijn met inachtname van de blokhiaten direct achter elkaar geplaatst, waarbij de plaats van een blok bepaald wordt door het overige. Magneetbanden lenen zich dan ook uitsluitend tot sequentiële bestandsorganisatie die met sequentiële verwerkingstechniek gebruikt wordt. Mutaties, weglatingen en aanvullingen zijn niet aan te brengen, aangezien men een tussengelegen blok niet opnieuw kan beschrijven. Muteren betekent dus copiëren onder gelijktijdige aanpassing van het nieuwe bestand (ga na dat door copiëren het opschuifprobleem niet voorkomt).
Magneetbanden vormen een betrekkelijk goedkoop medium en vooral voor opslag buiten de computer zijn magneetbanden eenvoudig, compact en goedkoop.
Men heeft in het verleden wel een enkel type magneetbandeenheid gemaakt waarbij wel sprake was van directe adressen en men ook wijzigingen kon aanbrengen. Aangezien deze typen betrekkelijk langzaam werkten is dit nooit een succes geworden, temeer omdat voor willekeurige toegang de wachttijden (enige minuten) toch te lang zijn. Indien door de aard van het probleem een sequentieel bestand met sequentiële verwerking geen bezwaar vormt, zijn magneetbanden nog altijd een aantrekkelijk goedkoop medium om mee te werken.

Door de technische noodzaak om bij mutaties een nieuw bestand op een andere magneetband te creëren krijgt men voor zeer weinig kosten en moeite bovendien een veiligheidsmaatregel tegen machinestoringen en fouten. Bewaart men namelijk enige generaties bestanden met tussengelegen mutaties (grootvader, vader, zoon systeem), dan heeft men een zeer effectieve methode om ingeval van calamiteiten de zaak weer te herstellen. Bij andere media, die op zich meer mogelijkheden bieden, kunnen de voorzorgen voor herstel na eventuele calamiteiten vaak zeer ingrijpend zijn.
Naast de magneetbanden bestaat een aantal andere vormen van secundair geheugen, die uit een oogpunt van bestandsorganisatie min of meer identiek zijn behoudens omvang en snelheid. Hiervan zijn te noemen trommel, schijven en magnetische strips.

Schijvengeheugens

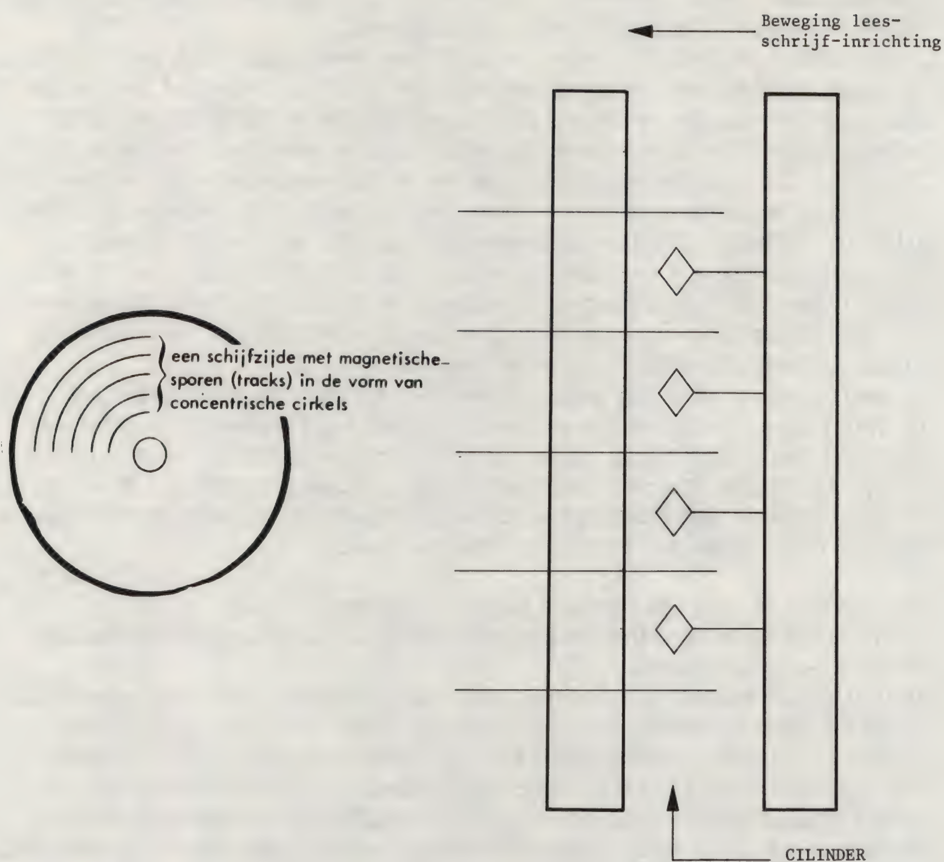
Ieder schijvenpakket bestaat uit een aantal schijven op één as, waarbij de schijven in de regel aan beide zijden in concentrische cirkels beschreven kunnen worden. Een dergelijke cirkel noemen we spoor of track (zie figuur 2). Het lezen en schrijven van en op een spoor gebeurt via een electromagnetische schrijfleeskop. Men kent daarbij twee systemen. In het eerste systeem is per schijfzijde slechts één schrijfleeskop aanwezig, die dan bevestigd is aan een verplaatsbare arm, zodat de kop voor ieder gebruik in de juiste positie boven een

spoor gebracht kan worden. Dit kost echter tijd. Hoewel er verschillende typen bestaan en bovendien de tijd ook afhangt over hoeveel sporen de arm zich moet verplaatsen, moet men voor een dergelijke insteltijd denken in de orde van grootte van 100 msec. Bij het tweede systeem zijn per schijfzijde evenveel schrijfleeskoppen als sporen aanwezig. In dit geval ontbreekt natuurlijk de insteltijd van de arm. Wel hebben we voor dit systeem evenals bij het eerste nog te maken met een wachttijd totdat de juiste positie van het spoor onder de schrijfleeskop is aangekomen. Dit ligt gemiddeld op een halve omwentelingstijd van de schijf. Dit betekent een grootteorde van 10 msec. Men kan deze tijd niet zo gemakkelijk veel kleiner maken, omdat men dan mechanische moeilijkheden krijgt met een te snel draaiende schijf. Een gedeeltelijke oplossing die men soms kiest is per spoor meerdere schrijfleeskoppen aanbrengen, maar dit verhoogt natuurlijk weer de kosten.
Schijven met vaste koppen zijn in het algemeen iets duurder. Bovendien zijn schijven met vaste koppen in vele gevallen niet uitwisselbaar.
Veel standaardprogrammatuur voor bestandsorganisatie is gebaseerd op de schijven met instelbare arm, met dien verstande dat men tracht tijdens het gebruik van bestanden deze armbewegingen zoveel mogelijk te beperken. Men beschouwt daartoe alle sporen van de verschillende schijven die recht onder elkaar liggen en dus achtereenvolgens bereikt kunnen worden zonder armverplaatsing als een geheel en noemt dit een c i l i n d e r (zie figuur 2). In tegenstelling tot magneetbanden is bij schijven wel sprake van adressering. Het adres van een blok wordt allereerst bepaald door het nummer van het schijvenpakket (indien althans meerdere pakketten zijn aangesloten), vervolgens door het cilindernummer en daarna door het spoornummer. Tot zover zijn vrijwel alle systemen gelijk behoudens verschillen in aantallen pakketten, cilinders en sporen. Tenslotte blijft over de positiebepaling per spoor. Hier bestaan meer principiële verschillen. In sommige systemen is ieder spoor bij voorbaat verdeeld in een aantal sectoren, waarbij per sector een blok opgeborgen kan worden. Adressering is dan simpel, omdat iedere sector een nummer heeft en via hardware na opgave van het gewenste sectornummer direct het juiste blok gelezen of geschreven kan worden. Deze werkwijze impliceert overigens min of meer een vaste bloklengte.
In andere systemen daarentegen wordt ieder spoor als een aaneengesloten opbergruimte beschouwd, waarvan alleen het beginpunt electronisch is gemarkeerd, en welke ruimte men dan nog naar eigen inzicht in blokken van bepaalde lengte kan indelen. Dit betekent dat men bij deze systemen althans per spoor weer terugvalt op een soort sequentiële organisatie, omdat de blokken vanaf het beginmerkpunt direct achter elkaar geplaatst zijn en de positie van ieder blok weer wordt bepaald door zijn voorganger. Dit zou betekenen dat het lezen van een blok het lezen van het complete spoor inhoudt. Bij het

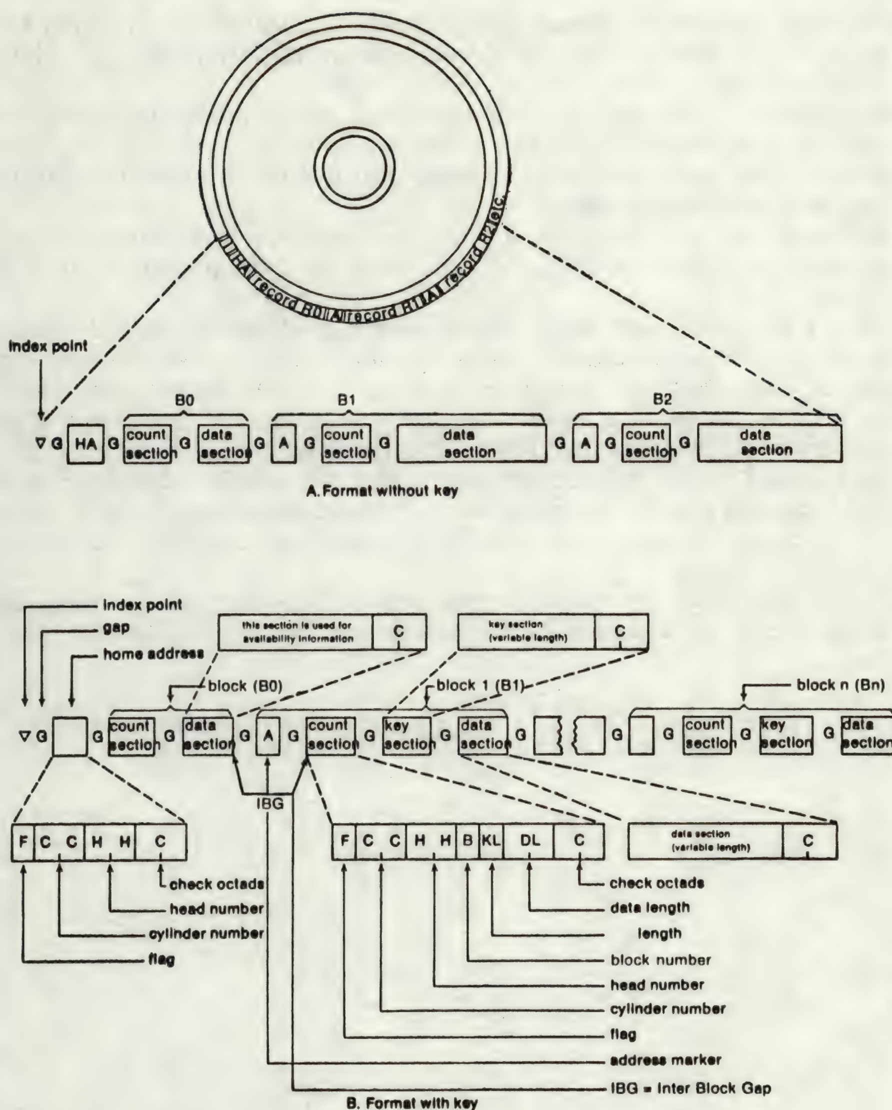
schrijven van een blok zou zelfs het spoor eerst ingelezen moeten worden in het primaire geheugen, dan het betrokken blok ingevuld, waarna het gehele spoor weer teruggeschreven moet worden. Deze complicatie wordt vermeden door extra faciliteiten in te bouwen in de electronische besturing van de schijven.
Ieder blok laat men voorafgaan door een zeer klein blok bestaande uit enige karakters, waarin een tellercode of een sleutel kan worden opgenomen. Soms zijn zelfs twee blokjes mogelijk waardoor zowel tellercode als sleutel gebruikt kunnen worden (zie figuur 3). Men kan dan aan de besturing van het schijvenpakket een opdracht geven om te zoeken naar een bepaalde tellercode en/of sleutel. De tijd die verloopt tijdens het passeren van het blokhiaat, tussen een dergelijk blokje en het daarop volgend blok met informatie, is voldoende voor de besturingselectronica om bij herkennen van de gewenste tellercode en/of sleutel het lezen of schrijven van het daaropvolgende informatieblok in te schakelen.
In wezen wordt dus het sequentieel zoeken per spoor overgelaten aan de hardware. In sommige uitvoeringen kan dit sequentieel zoeken via hardware zich zelfs uitstrekken over alle sporen van een cilinder. In het algemeen kunnen blokken op een schijvengeheugen herschreven worden zonder verstoring van overige informatie, mits natuurlijk de nieuwe bloklengte overeenkomt met de oude.

### Trommels en magnetische strips

Trommels en magnetische strips bieden na het voorgaande weinig nieuwe gezichtspunten.
Indien de trommel is uitgerust met minstens één kop per spoor kan de hele trommel beschouwd worden als één cilinder. Bij enkele trommels komen mechanisch verplaatsbare koppen voor. In dat geval worden alle sporen die bereikt kunnen worden in een bepaalde stand van de koppen als een cilinder beschouwd en de trommel bestaat dan uit evenveel cilinders als het aantal posities dat iedere kop kan innemen. Wachttijden liggen ook hier in de orde van 10 msec.
Ook bij magnetische strips wordt het cilinderconcept gehanteerd. Alle sporen die in een bepaalde stand van de koppen en via het omleggen van de strip om de trommel bereikt kunnen worden vormen dan weer een cilinder. Indien van strip gewisseld moet worden dient men rekening te houden met wachttijden in de orde van 500 msec.




Figuur 2: Een schijveneenheid bestaande uit verscheidene roterende schijven waarbij met behulp van verscheidene gelijk bewegende lees/schrijfkoppen steeds een'cilinder' met informatie beschikbaar is.




Figuur 3: Voorbeeld van een indeling van een spoor bij een schijvengeheugen zonder sectorindeling.

Bij figuur 3:

- Blok B1 kan de z.g. key sectie bevatten. Zij is bij sommige organisatiemethoden nodig om de data sectie te definiëren.
- Het index point is een markering van de onderste plaat van de disk om het begin van de track te kunnen identificeren.

- De gaps, tussen de verschillende blokken, stellen het systeem in staat om de positie van een blok in één omwenteling van de disk te identificeren.
- Elke track heeft een z.g. home address sectie norm ideal waarvan een track geïdentificeerd kan worden.
  In het home address zijn cilindernummer (CC) en schrijf-/leeskop nummer (HH) opgenomen.
- Het doel van blok B0 is om zorg te dragen voor een aantal administratieve funkties die nodig zijn bij het schrijven en lezen van een spoor.
- De track identificatie in de Count sectie is niet gerelateerd (zoals bij het home address) aan het begin van het volume maar aan het begin van de file. De track identificatie is uitgebreid met het bloknummer.
- De lengte van de data sectie van de verschillende blokken wordt gespecificeerd in de voorafgaande count sectie.
- De blokken B1 tot en met Bn verschillen van het B0 blok doordat zij een Address Marker bevatten en al of geen key sectie mogen bevatten. De address marker wordt door de Control Unit gebruikt ten behoeve van de hardware.
- De lengte van de key sectie wordt gespecificeerd in de voorafgaande count sectie. Als de key sectie niet is opgenomen dan is zijn lengte 0.

# 5 VEEL GEBRUIKTE OPSLAGSTRUCTUREN EN BIJBEHORENDE VERWERKINGSTECHNIEKEN

## 5.1 *Sequentiële bestandsorganisatie*

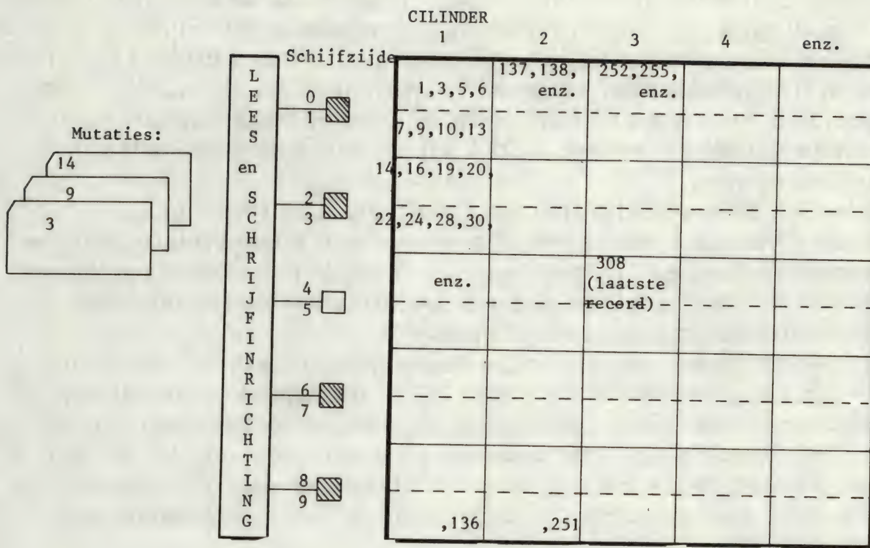
Bij sequentiële bestandsorganisatie zijn alle records fysiek achter elkaar geplaatst zonder adressering, zodat de positie van ieder record bepaald wordt door zijn voorganger (efficiënt geheugengebruik). Hoewel niet strikt noodzakelijk bedoelt men met deze organisatie meestal ook dat de records aan de hand van een sleutel in een bepaalde volgorde zijn gerangschikt.
Tengevolge van deze eigenschappen leent een sequentiële bestandsorganisatie zich praktisch alleen voor s e q u e n t i ë l e v e r w e r k i n g s t e c h n i e k (sequentieel access) hetgeen weer b a t c h v e r w e r k i n g (groepsgewijze verwerking) tot gevolg heeft. Immers voor een bewerking op één record moet men gemiddeld het halve bestand doorlopen. Daarom zal men veelal bewerkingen voor een aantal records verzamelen (mutatiebestand), deze op sleutelvolgorde sorteren en dan tegelijk verwerken. Door de directe aansluiting van records betekenen weglatingen en aanvullingen tevens een copiëring van het bestand.

M a g n e e t b a n d e n
Voor magneetbanden is deze organisatie praktisch de enige bruikbare. Aangezien daarbij wegens technische redenen een tussengelegen blok bovendien niet gewijzigd kan worden, impliceert dit voor iedere vorm van mutatie een copiëring van het bestand.
Bij sequentiële organisatie met sequentiële verwerking speelt s o r t e r e n veelal een belangrijke rol. Iedere keer dat men volgens een andere sleutel sequentieel wil verwerken moet het bestand eerst overeenkomstig die sleutel geordend worden.

Voorbeeld:
Stel we willen giro-overschrijvingen sequentieel verwerken. Men heeft dan een sequentieel bestand met records waarvan het rekeningnummer de sleutel is. Men verzamelt gedurende zekere tijd (een dag) alle mutaties en plaatst die op een band. Deze mutaties vormen dan ook een (tijdelijk) sequentieel bestand, zij het nog ongeordend. Dan volgt de eerste sorteergang voor dit tijdelijke bestand, waarbij de mutaties worden geordend volgens rekeningnummers van afschrijving.

Sequentieel kan dan deze mutatieband tezamen met het hoofdbestand verwerkt worden tot een nieuw voorlopig bestand, waarin alle afschrijvingen zijn verwerkt. Dan volgt de tweede sorteergang voor de mutaties, waarbij de volgorde wordt overeenkomstig rekeningnummers van bijschrijving. Dit mutatiebestand met het voorlopig nieuwe hoofdbestand worden dan sequentieel verwerkt tot het nieuwe hoofdbestand. Het is natuurlijk een gesimplificeerde voorstelling, want allerlei noodzakelijke nevenwerkzaamheden zijn hier buiten beschouwing gelaten zoals opbouw van het noodzakelijke uitvoerbestand, bestanden voor centrale registratie, controle, statistiek, enzovoorts. Wel is uit dit voorbeeld duidelijk dat sequentiële organisatie weinig moeilijkheden geeft bij beveiliging tegen calamiteiten. Immers in het voorgaande voorbeeld vernietigen we niet het oude hoofdbestand nadat het nieuwe is gevormd, maar bewaren dit benevens het mutatiebestand. Het oude hoofdbestand noemen we dan de vader en het nieuwe bestand de zoon. Bij verwerking van de volgende serie mutaties wordt de vader dan de grootvader, de zoon de vader en het dan nieuwe hoofdbestand de zoon.
Men heeft daarna drie generaties hoofdbestanden met tussengelegen mutaties. De daarop volgende keer kan men dan de oudste generatie vernietigen hetgeen eenvoudig gebeurt door de betrokken magneetbanden opnieuw te gebruiken. De beveiliging bestaat dus simpel uit het veilig bewaren van een aantal generaties hoofdbestanden met de daarbij behorende mutatiebestanden.

Schijvengeheugens
Bij de sequentiële bestandsorganisatie op schijven worden de informatie-eenheden direct achter elkaar vastgelegd en wel zodanig dat hogere identificaties (keys) ook hogere schijvenadressen hebben. Hierbij is geen enkele directe relatie tussen een recordidentificatie en het bijbehorend schijvenadres. Een schematische voorstelling van een dergelijk bestand is opgenomen in figuur 4. De nummers 1, 3, 5, 6, 7, 9 enzovoorts geven de records weer met de desbetreffende identificaties. Op elke schijfzijde worden in dit voorbeeld per informatiespoor steeds vier records vastgelegd. De ponskaartafbeeldingen met de nummers 3, 9 en 14 geven de mutaties weer die betrekking hebben op de records met de bijbehorende identificaties. Voor het verwerken van deze mutaties moet elke informatie-eenheid gelezen worden en in de computer worden vergeleken met de in de mutatiekaart opgenomen identificatie. Wanneer een gelijkheid wordt geconstateerd kan het desbetreffende record gemuteerd worden met de in de kaart opgenomen informatie. Om deze procedure zo efficiënt mogelijk te laten verlopen worden de mutaties vooraf gesorteerd in dezelfde volgorde als die waarin het bestand is vastgelegd. Per bewerking moet bij deze methoden wel een redelijke verhouding bestaan tussen het aantal mutaties en het aantal records in het bestand (batchverwerking).




Figuur 4: Schematische voorstelling van sequentiële bestands-organisatie.

Deze werkwijze maakt geen gebruik van de specifieke mogelijkheden die direct toegankelijke externe geheugens bieden en wordt dan ook het minst van alle methoden toegepast. Daarbij komt dat de eenmaal vastgelegde logische volgorde gehandhaafd dient te blijven. Daar we liever niet copiëren (hoeft niet bij schijven) ontstaat het probleem dat er binnen het bestand geen ruimte beschikbaar is voor het tussenvoegen van nieuwe records. Dit wordt meestal opgelost door de nieuwe records tijdelijk in een ander gebied in de schijveneenheid op te nemen. In een later stadium worden deze records dan weer samengevoegd met het oorspronkelijke bestand. Teneinde te allen tijde een normale verwerking van het gehele bestand mogelijk te maken, wordt met behulp van verwijsadressen (wijzers) de logische volgorde gehandhaafd. Wanneer bijvoorbeeld in het bestand weergegeven in figuur 4 een nieuw record met identificatie 8 moet worden vastgelegd, kan dit record bijvoorbeeld in cilinder 4 worden geplaatst, terwijl in record 7 een verwijzing naar record 8 wordt opgenomen. Record 8 krijgt dan op zijn beurt weer een verwijzing naar record 9 in cilinder 1. Op deze wijze blijft voor de verwerking de numerieke volgorde binnen het gehele bestand gehandhaafd.

## 5.2 *Index-sequentiële bestandsorganisatie*

Een vaak gebruikte tussenvorm van een sequentiële en een willekeurig toegankelijke opslagstructuur met een tabelrelatie tussen recordsleutel en recordadres is de zogenaamde index-sequentiële organisatie (geïndiceerd sequentiële organisatie). Bij veel toepassingen heeft men namelijk behoefte aan een opslagstructuur waarbij:
- records willekeurig toegankelijk zijn (zodat magneetbanden niet bruikbaar zijn);
- bestanden zo groot zijn dat ook trommels niet voldoen;
- er soms behoefte aan is om alle records in sleutelvolgorde te verwerken, terwijl er eigenschappen van het bestand zodanig zijn dat willekeurig toegankelijke opslagstructuren niet in aanmerking komen (door omvang en sleutelverdeling).

Aangezien zodoende alleen magneetschijven in aanmerking komen heeft men een organisatie ontwikkeld die aangepast is aan de eigenschappen van een schijf. Met name aan het feit dat bij een bepaalde stand van de schijfarm vele sporen op de schijven snel uitgelezen kunnen worden (ze 'vormen een cilinder') maar dat een verandering van de stand van de schijfarm relatief lang duurt (minimalisatie armbeweging).

De index-sequentiële organisatie komt dan hierop neer dat vanaf een vaste plaats (veelal cilinder 0), een zogenaamde cilindertabel (cilinderindex) in het kernengeheugen wordt gebracht en dat hieruit afgelezen kan worden wat voor iedere cilinder de hoogst voorkomende sleutel is. De cilindertabel is op sleutelvolgorde gesorteerd zodat met een binaire zoekmethode snel gevonden kan worden in welke cilinder een record met een bepaalde sleutel zich kan bevinden (hoewel niet strikt noodzakelijk zal men mutaties op zo'n bestand liefst gesorteerd verwerken, omdat dan het aantal armbewegingen minimaal is!).

Van een vast spoor op iedere cilinder (veelal spoor 0 van een cilinder) wordt dan de zogenaamde sporentabel (tracktable, spoorindex, trackindex) in het kerngeheugen gelezen. Hierin staat weer in sleutelvolgorde vermeld wat voor ieder spoor de hoogst voorkomende sleutel is. Na met een zoekmethode zo voor een bepaalde recordsleutel vastgesteld te hebben in welk spoor zich het betreffende record kan bevinden, kan dit spoor naar het kernengeheugen gebracht worden (we nemen hierbij dus stilzwijgend aan dat er in het kernengeheugen voldoende ruimte is om een compleet spoor in te lezen; is dit niet het geval dan moeten in de sporentabel ook de nummers van de spoordelen worden opgenomen). Met een sequentiële zoek kan dan in het spoor het goede record gevonden worden.

Bij een multivolume file wordt ook vaak nog een volume-index opgebouwd, die de hoogste sleutel van elk volume aangeeft. Aldus krijgt men indices van drie niveaus: volume-index, cilinder-index, spoorindex.

Samenhangend met het feit dat in een gegeven bestand records toegevoegd of verwijderd worden, bestaan voor de implementatie van deze bestandsvorm verschillende varianten:

- De simpelste vorm, alleen te gebruiken als er weinig records aan het bestand worden toegevoegd, is met de gesorteerde records bestand en tabellen op te bouwen op de beschikbare sporen.
- Een variant is dat men bij de eerste opbouw van het bestand de sporen van een cilinder opeenvolgend met records vult behalve één of meer overloop-sporen (overflow) van die cilinder. Moet later bij een mutatie een record tussengevoegd worden, dan gebeurt dit na op de goede plaatsen de aanwezige records een plaats opgeschoven te hebben. Het record dat zodoende van het 'primaire spoor afvalt', wordt dan op de eerstvolgende vrije plaats van een overloopspoor geplaatst. Een nadeel van deze methode is dat via het kernengeheugen geschoven en in het overloopgebied nog eens sequentieel gezocht moet worden, en dat de sporentabel moet worden bijgewerkt.
- Een tweede variant beoogt het inlezen van een overloopspoor zolang mogelijk uit te stellen door in eerste instantie de primaire sporen niet voor 100% te vullen. Pas wanneer de vrije ruimte op een spoor opgebruikt is, moeten overloopsporen ingelezen worden. Aangezien het 'opgebruiken' van vrije ruimte als regel niet gelijkmatig is voor alle sporen, is deze variant niet zo gunstig ten aanzien van het geheugengebruik. Als bij de eerste variant moet mogelijk de sporentabel bijgewerkt worden.
- In een derde variant worden de sporen weer voor 100% gevuld, maar bovendien bevat ieder record in een primair spoor zo nodig een wijzer naar het begin van een gesorteerde ketting in een overloopspoor. Hiermee is het opschuiven van records in een primair spoor overbodig en is de sequentiële zoek in een overloopspoor efficiënter dan bij de eerste variant. Het bijwerken van de sporentabel is niet nodig.

Het verwijderen van records gebeurt als regel niet door een fysieke verwijdering, maar door het aanwezige record van een kenmerk (flag) te voorzien. Bij alle drie varianten zal het nodig zijn om het bestand periodiek te reorganiseren, namelijk wanneer er door een hoge file turnover te veel records in overloopsporen zitten. Reorganiseren gebeurt eenvoudig door sequentieel overschrijven op een nieuwe schijf met gelijktijdig leegmaken van de overloopsporen en vervallen records. Uiteraard worden hierbij nieuwe cilinder- en sporentabellen opgebouwd.

Samenvattend kan men stellen:

- De index-sequentiële organisatie neemt meer ruimte in beslag dan de sequentiële:

  De cilinder- en track-index moeten opgeslagen worden; dit is vaak 5 tot 10% van het geheel.

Voorts is het primaire gebied groter dan het bestand vraagt teneinde nieuwe records in de gaten te kunnen plaatsen (zie varianten). Daarnaast dient men overflowruimte te reserveren voor het geval de gaten vol lopen.
- Naarmate men start met een hogere bezettingsgraad (weinig gaten) van het primaire gebied zal men met meer overflow (overloop) moeten rekenen.
- Naarmate meer indices (bijvoorbeeld volume-index en cilinderindex) in het kerngeheugen passen, neemt het aantal armbewegingen om een record te bereiken af (maximaal 3, minimaal 1).

Voorbeeld:
De algemene opbouw van een indexed sequential bestand is weergegeven in figuur 5. De cilindertabel is hier opgenomen in een deel van cilinder 01 en bevat de hoogste identificatie en het laagste adres van elke cilinder. Wanneer nu een mutatie binnenkomt met bijvoorbeeld identificatie 279, wordt het lees/schrijfmechanisme eerst verplaatst naar de cilinder 04. Immers 279 is hoger dan de hoogste identificatie van cilinder 03, maar lager dan die van cilinder 04. Aangezien in de cilindertabel bij de hoogste identificatie van cilinder 04 (612) ook het laagste adres binnen diezelfde cilinder is opgenomen, kan nu een opdracht aan het lees/schrijfmechanisme worden gegeven om van cilinder 01 naar cilinder 04 te gaan.
In figuur 6 is de inhoud van cilinder 04 en de bijbehorende indeling van de track-index weergegeven. Op elke track binnen cilinder 04 zijn steeds drie records opgenomen. Voor het lokaliseren van de track waarop het record met identificatie 279 voorkomt, wordt nu de track-index geraadpleegd. Hieruit blijkt dat het betrokken record zich bevindt op adres 043. Immers 279 is hoger dan 117 en 198, maar lager dan 309. Nu kan de juiste schijfzijde binnen de cilinder geactiveerd worden. Het gewenste record binnen die track wordt bereikt als tijdens een aantal vergelijkingen van mutatie- en recordidentificaties een gelijkheid wordt gesignaleerd.
De procedure die gevolgd wordt bij het opnemen van nieuwe records in een reeds eerder vastgelegd bestand is weergegeven in de figuren 6 en 7. Volledigheidshalve geven wij eerst een toelichting op figuur 6, speciaal met betrekking tot de track-index. Deze index bestaat uit twee delen, een 'normal-entry' waarin wordt bijgehouden hoe de werkelijke situatie is binnen een track, en een 'overflow-entry', waarin wordt bijgehouden hoe de logische track-samenstelling is nadat een aantal nieuwe records is toegevoegd. Voordat het eerste nieuw op te nemen record moet worden verwerkt, hebben de normal-entry en de overflow-entry dezelfde inhoud bestaande uit de hoogste identificatie (key) van die track en het startadres (T-A). Stel nu dat twee nieuwe records moeten worden opgenomen, die de identificaties 142 en 450 hebben. Aangezien altijd binnen een track de logische volgorde wordt gehandhaafd, wordt record 198 van track 042 'afgeschreven' en geplaatst in de gereserveerde ruimte (overflow-gebied)

op track 048. Iets dergelijks gebeurt ook met record 516 voor het opnemen van het nieuwe record 450.
Daar de inhoud van de normale tracks nu gewijzigd is, wordt ook de normal-entry van de track-index aangepast. Zo worden de identificaties 198 en 516 respectievelijk vervangen door 192 en 450. De track-adressen blijven ongewijzigd.
In de overflow-entry blijven de identificaties ongewijzigd, wel worden de adressen aangepast. Dit is logisch daar de records 198 en 516 zijn verplaatst naar track 048. De aanduidingen 048-1 en 048-2 geven aan dat de bijbehorende records respectievelijk de eerste en de tweede zijn binnen track 048. Het voordeel van deze aanpak is dat nieuwe records worden opgenomen in de cilinder, waarin ze volgens de logische opbouw van het bestand ook thuis horen.

INDEXED SEQUENTIAL BESTANDSORGANISATIE

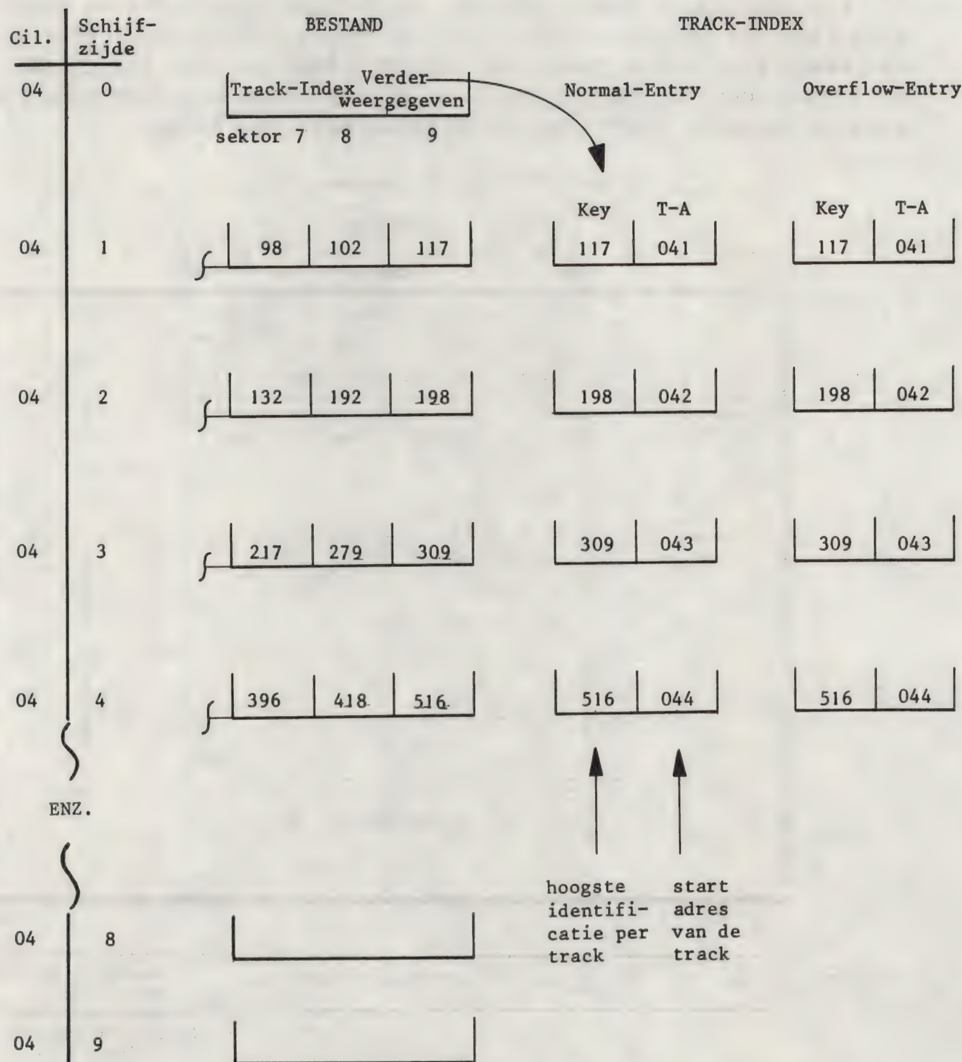
| | CILINDER | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | ← | BESTAND | | → | | |
| schijfzijde | 01 | 02 | 03 | 04 | 05 | 06 | 07 | enz. |
| 0 | C I | | track-index | track-index | track-index | track-index | | |
| 1 | L I N | | 1,4,16 | 98,102, 117 | 614,638, 662 | 875,878, 879 | | |
| 2 | D E R | | 19,25,38 | 132,192, 198 | Enz. | Enz. | | |
| 3 | Index | | Enz. | 217,279, 309 | | | | |
| 4 | | | | 396,418, 516 | | | | |
| 5 | | | | Enz. | | | | |
| 6 | | | | | | | | |
| 7 | | | ,91 | ,612 | ,873 | ,999 | | |
| 8 | | | GERESERVEERD VOOR | | | | | |
| 9 | | | NIEUWE RECORDS | | | | | |

Cilinder-index: * 91 (03); * 612 (04); * 873 (05); * 999 (06);

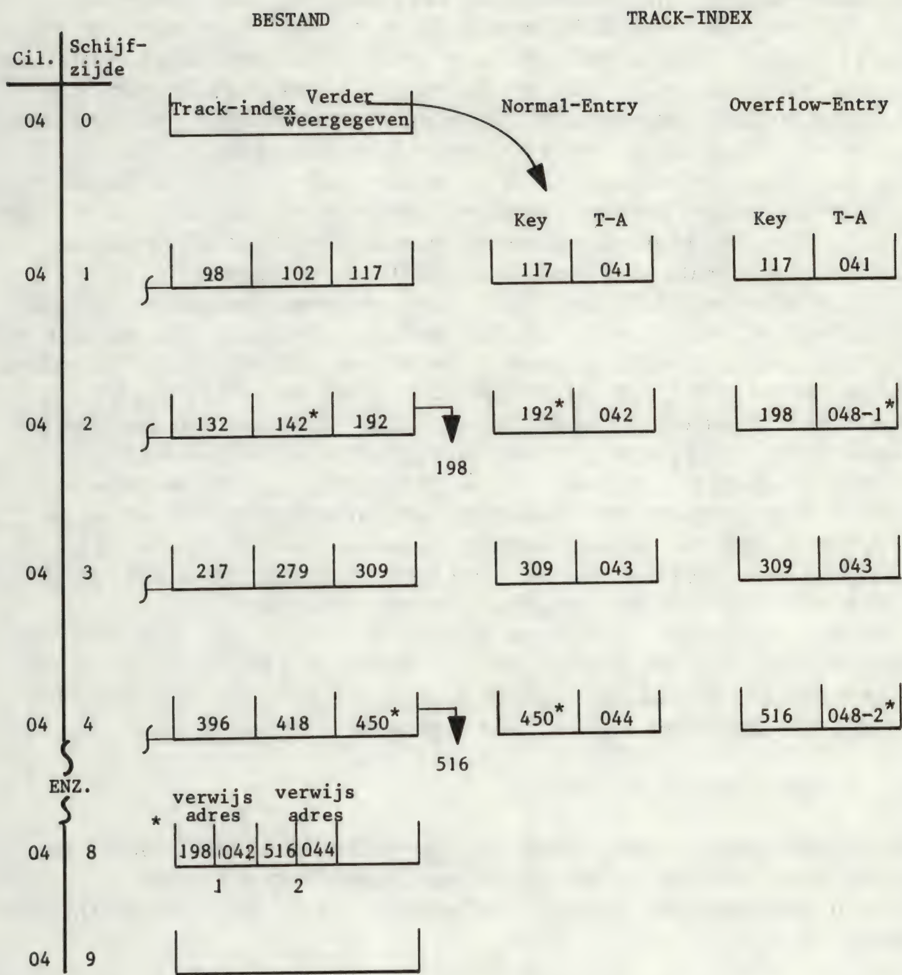
* hoogste identificatie in cilinder

() laagste adres in cilinder

Figuur 5: Schematische voorstelling van een indexed-sequential bestand.




Figuur 6: Situatie direct na het vastleggen van het bestand.




Figuur 7: Situatie direct na het toevoegen van records met de identificaties 142 en 450 (wijzigingen t.o.v. figuur 6 zijn aangegeven met *).

## 5.3 *Directe bestandsorganisatie*

Bij de directe of willekeurig toegankelijke bestandsorganisatievorm is er, in tegenstelling tot de in het voorgaande beschreven methoden, wel een relatie tussen de identificaties en de adressen van de vastgelegde gegevens. Hierdoor is een snellere toegang tot ieder record mogelijk. In het algemeen worden de vastgelegde gegevens bij deze methode niet eerst in een bepaalde volgorde vastgelegd. Het voordeel hiervan is dat een sorteerbewerking ten behoeve van het verwerken van het bestand naar een vooraf vastgesteld gezichtspunt komt te vervallen. De methoden van adresbepaling van de records kunnen onderverdeeld worden in twee groepen: de directe adressering en de indirecte adressering.
Bij directe adressering is er voor elke mogelijke sleutelwaarde een adres gereserveerd en dus voor elk mogelijk record opslagruimte. Dit impliceert dat het aantal mogelijke sleutels niet groter mag zijn dan het aantal beschikbare adressen.
Bij indirecte adressering kan het aantal mogelijke sleutels wel groter zijn dan het aantal beschikbare adressen. De adresberekeningsalgoritme zal dan ook aan meerdere sleutels hetzelfde adres kunnen toewijzen. Deze sleutels noemt men synoniemen.
Naast dit essentiële verschil tussen deze beide vormen van adresbepaling uit een sleutel, zijn er nog praktische verschillen:

- Directe adressering beoogt zo snel mogelijk het adres te bepalen. Dit betekent dat de recordsleutels bij voorkeur numeriek moeten zijn en enigszins op de geheugenadressen moeten lijken. Bij indirecte adressering bestaat deze beperking niet en kunnen de sleutels volledig door de gebruiker bepaald worden.
- Indien de feitelijk voorkomende sleutels slechts een deel zijn van de verzameling mogelijke sleutelwaarden dan ontstaan bij directe adressering veelal grote gaten (inefficiënt geheugengebruik); bij indirecte adressering hoeft dit niet op te treden.

### 5.3.1 Directe adressering

Bij directe adressering bevat de informatie die in het bestand verwerkt moet worden het adres van het bijbehorende record.
Binnen deze methode is weer een aantal verschillende mogelijkheden aanwezig:

Schijfadres = sleutel
Allereerst is het mogelijk de identificaties zelf als schijvenadres te gebruiken. Dit kan echter niet altijd, daar de toegepaste identificaties meestal zijn opgebouwd volgens een codeersysteem dat geen aansluiting geeft op de numerieke adressen van de schijvengeheugens. Is dit wel het geval, dan moet gelet worden op de compactheid. Wanneer bijvoorbeeld de identificaties liggen binnen de waarden

00000 en 99999 en er slechts 10000 nummers uit deze groep worden gebruikt dan leidt het hanteren van deze vorm van adressering tot een zeer inefficiënt gebruik van de capaciteit van het schijvengeheugen.
Het probleem kan worden opgelost, door de gebruikte codering zodanig om te zetten, dat de nieuwe codering wel overeenkomst vertoont met de schijvenadressen. In veel gevallen zal dit echter op grote praktische bezwaren stuiten.

Schijfadres = sleutel via tabel
Wanneer recordidentificaties en schijvenadressen niet identiek zijn of gemaakt kunnen worden, is er nog een tweede mogelijkheid om tot directe adressering te komen. Dit geschiedt door naast de identificaties in de informatie van bestandsmutaties het diskadres als extra gegeven op te nemen. Een eis bij deze mogelijkheid is wel dat vooraf voor elke identificatie een diskadres wordt vastgesteld en dat dit adres steeds via een voorbewerking als extra gegeven wordt opgenomen in de bestandsmutaties.
De computer krijgt - wanneer aan deze eis is voldaan - naast de nieuwe informatie ook het adres van het bijbehorende record ingevoerd voor het bijwerken van het bestand. In het bijzonder voor het bijwerken van statische bestanden is deze methode aantrekkelijk.
Een betere oplossing, waarbij nog meer geautomatiseerd is, wordt verkregen door de eenmaal vastgestelde relatie tussen de identificaties en de schijvenadressen in de vorm van een tabel op te nemen. Hierdoor wordt echter wel de effectieve capaciteit van het externe geheugen aangetast. Bovendien zijn tijdens de verwerking van deze bestanden steeds twee bewegingen van het lees/schrijfmechanisme nodig. Te weten: één om de tabel te raadplegen en daarna één om het juiste record te bereiken. Dit is minder aantrekkelijk omdat de zeer snelle toegang tot elk afzonderlijk record juist één van de belangrijkste voordelen is bij het gebruiken van random bestandsorganisatie.
Een verbetering van de snelheid kan worden bereikt door zowel voor het bestand als voor de adrestabel een afzonderlijke lees/schrijfinrichting te gebruiken. Bij een zeer geïntegreerde verwerkingsprocedure van een groot aantal bestanden is deze werkwijze slechts bij uitzondering mogelijk. De gewenste snelheidsverbetering kan ook nog bereikt worden door gedurende de gehele computerbewerking de directe adresseertabel in het werkgeheugen van de computer op te nemen. Hiervoor dient men wel over voldoende interne geheugencapaciteit te beschikken. Het is duidelijk dat deze methode afhankelijk is van de computerconfiguratie, de omvang van de bestanden en de bijbehorende verwerkingsprocedures.

Schijfadres via relatieve adressering
Een, de laatste tijd steeds meer gebruikte vorm van directe adressering, is de relatieve adressering.

Dit verklaren we aan de hand van een eenvoudig voorbeeld.
Neem aan dat we een bestand hebben verdeeld over 5 schijfpakketten (volumes), ieder volume omvat 200 cilinders, per cilinder zijn 18 sporen in gebruik en de recordlengte is van dien aard dat per spoor 13 records opgeborgen kunnen worden. Het gehele bestand omvat dan 5 x 200 x 18 x 13 = 234000 records. Om een bepaald adres aan te geven is echter een getal van 8 cijfers nodig, bijvoorbeeld 31751411, hetgeen betekent volume 3, cilinder 175, spoor 14 en sector of positie 11. We kunnen de records echter ook doorlopend nummeren, in dat geval van 0 tot en met 233999. Dit noemt men dan een relatief adres. Tussen relatief en absoluut adres bestaat echter een éénduidig verband.
Aangenomen dat ook volume-, cilinder-, spoor- en sectoradressering op 0 beginnen rekent men het gegeven voorbeeld van 31751411 als volgt om naar een relatief adres ((3 x 200 + 175) x 18 + 14) x 13 + 11 = 181543.
Precies in omgekeerde volgorde rekent men bij omzetting van relatief adres naar absoluut adres. Bijvoorbeeld het relatief adres 112028. We delen eerst door 13, daarna het quotiënt door 18 en vervolgens het quotiënt door 200.
Zo is 112028 = 8617 x 13 + 7
8617 = 478 x 18 + 13
478 = 2 x 200 + 78.
Het absolute adres is dus 20781307 namelijk volume 2, cilinder 78, spoor 13 en sector 7.
Gezien de rekensnelheid van moderne computers vormen dergelijke omrekeningen geen enkel probleem.
De omzetting van sleutel naar adres kunnen we in twee stappen denken namelijk eerst van sleutel naar relatief adres en daarna de omzetting tot absoluut of fysisch adres. In het volgende houden we ons alleen bezig met de eerste omzetting. De eenvoudigste transformatie is één op één, dat wil zeggen sleutel = relatief adres of sleutel = relatief adres + vast bedrag. Dit laatste kan voorkomen om nullen aan de voorzijde van een sleutel te vermijden. Bijvoorbeeld men heeft een bestand van 8000 records met relatieve adressen 0 tot en met 7999 en nummert dan de sleutels van 1000 tot en met 8999. Dit systeem geeft enerzijds het grote voordeel dat geen synoniemen optreden en de beschikbare geheugenruimte optimaal kan worden benut. Anderzijds bestaat het grote nadeel dat de sleutels worden bepaald door het computersysteem en dus niet gekozen kunnen worden vanuit gebruikersstandpunt. Voor vele toepassingen is dit een ontoelaatbare beperking. Heeft men als voorbeeld een bestand waarbij de records bestaan uit persoonsgegevens terwijl de sleutel gevormd wordt door de naam, dan kan men niet de naam van de mensen wijzigen en hen als 'naam' een relatief adresnummer geven.

S c h i j f a d r e s via relatief adres, relatief adres via tabel

Een oplossing die toch van deze methode gebruik maakt bestaat hieruit dat men buiten de computer een soort naamlijst aanlegt waarin achter ieders naam het bijbehorende relatieve adres voorkomt. Iedere keer dat men aan de computer mutatiegegevens toevoert, moet men tevens de opgezochte 'sleutel' uit de naamlijst meegeven. Het expliciet opzoeken is soms te vermijden, wanneer men werkt met voorbedrukte of voorgeponste formulieren bijvoorbeeld een girokaart, waarop niet alleen de naam, maar ook het gironummer voorkomt.
Op een bankcheque kan men behalve de naam en het gebruikelijke banknummer ook reeds een dergelijk relatief adres vermelden. Nog een stap verder is de 'naamlijst' zelf in het computergeheugen opgenomen. Gezien de omvang van een dergelijke lijst kan die veelal niet in het primaire geheugen staan, dat wil zeggen deze naamlijst wordt een bestand op zichzelf waarvoor men bijvoorbeeld sequentieel of index-sequentieel kan kiezen. De records van een dergelijk bestand zijn natuurlijk erg eenvoudig want ze bestaan in principe uit slechts twee velden, namelijk de sleutel en het relatief adres van het record in het directe bestand. De toegang tot een record betekent dan toch reeds minimaal twee verschillende achtereenvolgende toegangen tot het achtergrondgeheugen. De omzetting van relatief adres naar een absoluut adres wordt door de meeste standaardprogrammatuur automatisch uitgevoerd, hierbij lettend op de plaats waar het relatief bestand zich bevindt op de schijf, de lengte van de fysische records, etc.
Met de laatste behandelde vorm van relatieve adressering is het eigenlijk dubieus of men nog wel mag spreken van relatieve adressering want het principe was dat de sleutel gelijk is aan het relatief adres.

## 5.3.2 Indirecte adressering

Werden in het voorgaande de sleutelwaarden toch beïnvloed door de mogelijkheden van de adreswaarden, bij indirecte adressering is deze koppeling niet meer aanwezig. In het algemeen is het aantal mogelijke sleutels veel groter dan het aantal gebruikte en het aantal beschikbare adressen. Het mogelijke sleutelgebied moeten we via de adresberekening (sleuteltransformatie) afbeelden op het beperkte ter beschikking staande relatieve adresgebied.
Dit proces wordt genoemd r a n d o m i z i n g (sleutelconversie procedure, sleuteltransformatie procedure) en gaat gepaard met het probleem van s y n o n i e m e n. Hoewel deze randomizing van het grootste belang is voor het slagen of falen van deze bestandsorganisatie, is hierover relatief weinig informatie te verstrekken. Het is ook niet mogelijk één of enkele algemeen bruikbare methoden aan te geven,

omdat de optimale keuze afhankelijk is hoe de gebruikte sleutels verdeeld zijn over het gebied van mogelijke sleutelwaarden.

Voorbeeld:
Neem aan een bestand van ongeveer 8000 records. De sleutel van deze records in een artikelnummer dat bestaat uit 8 decimale cijfers. Als primair gebied wordt een geheugenruimte voor 10000 records ter beschikking gesteld en daarenboven een zekere geheugenruimte voor overloop. Dat wil dus zeggen dat wij bij voorbaat reeds uitgaan van een bezettingsgraad van het primaire gebied van ongeveer 80%.
Bezien we nu de sleutel. Hiervan is alleen gegeven dat deze bestaat uit een getal van 8 decimale cijfers. Zonder verdere toelichting betekent dit dat alle mogelijke sleutels gevormd worden door de getallenrij van 0 tot $10^8-1$. De 8000 gebruikte sleutels zijn echter niet zomaar een greep uit de $10^8$ mogelijkheden. Door vermelding van het feit dat de sleutels hier artikelnummers betreffen kunnen we reeds vermoeden dat in iedere sleutel van 8 cijfers enkele codes verwerkt zijn zoals bijvoorbeeld fabrieksnummer, fabrikage methoden, prijsindex, enz. Het gevolg van deze op zich misschien wel logische indeling van artikelnummers is, dat in de verdeling van de artikelnummers over de gehele getallenrij van 0 tot $10^8$ hier en daar verdichtingen en verdunningen optreden. Zouden we hier geen rekening mee behoeven te houden dan zouden we in dit geval als eenvoudige transformatie kunnen nemen: laat de voorste 4 cijfers weg en beschouw de laatste 4 cijfers als relatief adres. Met evenveel recht hadden we de voorste 4 cijfers als relatief adres kunnen beschouwen en de laatste 4 cijfers kunnen weglaten.
Het is duidelijk dat met een dergelijke transformatie, ook zonder statistische berekeningen, bij ongelijke verdeling van sleutelwaarden ook ongelijke verdelingen van records over het primair toegewezen gebied zullen plaatsvinden.
Op bepaalde plaatsen houdt men ruimte over, op andere daarentegen komt men tekort en dit veroorzaakt de o v e r l o o p.

Randomizing heeft tot doel de omrekening van sleutelwaarden naar adressen zodanig te definiëren, dat ondanks bepaalde systematiek in de sleutelsamenstelling, de verdeling van alle records over de beschikbare gestelde ruimte statistisch gezien zo gelijkmatig mogelijk wordt.
In hoeverre in het hiervoor gegeven voorbeeld het weglaten van de voorste 4 cijfers een juiste methode is zou alleen een nauwkeurige statistische analyse betreffende het voorkomen van sleutelwaarden kunnen leren. Het is mogelijk dat dit in sommige gevallen een prima bruikbare methode is.
Welke berekeningsmethode het beste kan worden toegepast, wordt bepaald door de efficiëntie waarmee de beschikbare externe geheugencapaciteit wordt gebruikt en het aantal daarbij voorkomende synoniemen.

Adresseringstechnieken
Vele adresberekeningstechnieken die bij indirecte adressering gebruikt worden, interpreteren de bit representatie van de sleutel als een binair getal of herleiden dit tot een binair getal.
Dit zou de eerste stap van de adresberekening kunnen worden.

Onderstaande technieken zijn voor het gemak alleen beschreven voor numerieke sleutels. Doel van elke goede adresberekening is een gelijkmatige spreiding van sleutels over adressen te bereiken. Voordat een bepaalde adresberekening gekozen wordt, zal dan ook een studie (statistische analyse) gemaakt moeten worden van de verdeling van feitelijke sleutelwaarden over het totaal aantal mogelijke sleutels, daarnaast van de opbouw van elke sleutel, terwijl men bovendien rekening zou moeten houden met de toekomstige ontwikkeling van het bestand (welke sleutels komen erbij, welke zullen verdwijnen?).
Een van de hulpmiddelen om de opbouw van een sleutel te analyseren is een cijfer analyse tabel. Deze tabel wordt vervaardigd door het tellen van de frequentie, waarmee de cijfers 0 tot en met 9 voorkomen in elke positie van bijvoorbeeld een artikelnummer. Een analyse van deze tabel geeft informatie met betrekking tot de verdeling van de cijfers in elke controlepositie. Deze analyse is een waardevol hulpmiddel bij het kiezen van een bepaalde techniek bij een bepaalde serie getallen.
Bijvoorbeeld: 16045 artikelen - controleveld (sleutel) van 8 posities.

| cijfer | positienummer in controlegetal | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 16045 | 1852 | | | 1574 | 1807 | 1738 | 5168 |
| 1 | | 3147 | 4408 | | 1652 | 2120 | 1748 | 5638 |
| 2 | | 1174 | 3792 | | 1587 | 1745 | 1743 | 4958 |
| 3 | | 2725 | 2231 | | 1620 | 1684 | 1610 | 281 |
| 4 | | 1194 | 2459 | | 1647 | 1378 | 1617 | |
| 5 | | 1267 | 3155 | | 1580 | 1647 | 1688 | |
| 6 | | 1234 | | 2198 | 1538 | 1560 | 1606 | |
| 7 | | 1228 | | 567 | 1560 | 1329 | 1450 | 1560 |
| 8 | | 1227 | | 1195 | 1630 | 1415 | 1411 | |
| 9 | | 989 | | 12076 | 1657 | 1360 | 1434 | |

Figuur 8.

De volgende voorbeelden geven enige methoden en technieken aan, welke kunnen worden gebruikt om een sleutel tot een diskadres om te rekenen.
Deze voorbeelden pretenderen niet een volledige serie formules te zijn, maar een leidraad tot die technieken, welke een aanvaardbaar resultaat opleveren.
Een verdelingsanalyse moet in ieder geval worden uitgevoerd op het gehele bestand vóórdat een bepaalde formule wordt gekozen. De formule moet worden toegepast op ieder artikelnummer van het bestand in plaats van steekproeven te nemen.

A. Extracting
Gebruik een gedeelte van de bestaande sleutel als basis voor diskadres.
Bijvoorbeeld:

| | | |
|---|---|---|
| 24 | W | 356 |
| 53 | TP | 4152 |
| 53 | T | 4149 |
| 30 | T | 1441 |
| | MW | 5092 |
| 48 | T | 773 |
| 78 | T | 3560 |
| 45 | TP | 4959 |
| 67 | TN | 12 |

Dit nummer is onderverdeeld op de volgende manier:
a. Prefix - 2 tekens
b. Afd.code - 0 - 1 - of 2 tekens
c. Artikelcode - 2 - 3 - of 4 tekens.
Indien men nullen toevoegt, waar een teken ontbreekt, ontstaat het volgende:

| | | |
|---|---|---|
| 24 | W0 | 3560 |
| 53 | TP | 4152 |
| 53 | T0 | 4149 |
| 30 | T0 | 1441 |
| 00 | MW | 5092 |
| 48 | T0 | 7730 |
| 78 | T0 | 3560 |
| 45 | TP | 4959 |
| 67 | TN | 1200 |

Aangenomen, dat de cijfer analyse tabel een regelmatige spreiding aangeeft voor het artikelnummer, dan zou dit wellicht kunnen dienen als basis voor een diskadres.

B. Folding (knippen)
Optellen of omwisselen van het ene gedeelte van de sleutel met het andere.
De cijfer analyse tabel gaf bijvoorbeeld de volgende resultaten:

a. Artikelnummer: Goede verdeling op de drie linkerposities. Slechte verdeling op de rechterpositie.
b. Afdelingscode : Zeer goede verdeling van het numerieke gedeelte van de tientallen positie.
c. Prefix : Goede verdeling van eenhedenpositie.

Deze verdeling zou tot de volgende methode kunnen leiden:

```
1.      24    WO    3560
2.      3564 = Basis voor diskadres.
```

Het voordeel van zowel de extracting als de folding methode is, dat de berekening niet erg ingewikkeld is.

C. Vermenigvuldiging (midsquaring)
De meest eenvoudige methode is kwadrateren en de middelste cijfers van het produkt als basis voor diskadres te gebruiken.
Bijvoorbeeld: 24WO3560$^2$ =

```
         24603560          (neem voor WO 60)
         24603560
        147621360.
       123017800..
        73810680...
     147621360.....
      98414240......
     49207120.......
0605335164673600
      5164 = Basis voor diskadres.
```

Deze methode kan meestal verbeterd worden door het gevonden getal van vier cijfers te vermenigvuldigen met het artikelnummer. Hierdoor zal het aantal duplicaten, welke door het kwadrateren zijn ontstaan, verminderen.
Bijvoorbeeld: 5164 x 24WO3560
In sommige gevallen zal het kwadrateren van het controlegetal in zijn geheel niet het beste resultaat opleveren. Als bijvoorbeeld is gebleken, dat de vier rechtse posities de beste spreiding vertonen, zullen zij bij deze techniek hun betekenis verliezen. Indien deze vier cijfers met de folding methode naar het midden van het controlegetal worden gebracht, zal hun betekenis veel meer tot uiting komen.

D. Deling
De deelmethode houdt in, dat de sleutel wordt gedeeld door een constante. De rest wordt gebruikt als diskadres.
De deler is meestal:

a. Het aantal bestandsrecords, welke zijn toegestaan voor het aantal posten in het hoofdbestand.
b. Het dichtstbijzijnde priemgetal bij het aantal uit punt a.

Het gebruik van een priemgetal wordt aangeraden, omdat dit alleen kan worden gedeeld door het cijfer één en door zichzelf. De kans, dat men vanuit verschillende controlegetallen dezelfde rest verkrijgt is bij het delen door een priemgetal veel minder groot.
Ook een deler, welke eindigt op de cijfers 1 - 3 - 7 of 9 kan goede resultaten geven.
Bijvoorbeeld:

```
10.000/24603560/2460
       20000
        46035
        40000
         60356
         60000
          3560 = Basis voor diskadres.
```

Het dichtstbijzijnde priemgetal bij 10.000 is 9973.
Hiermee krijgen we:

```
9973/24603560/2467
     19946
      46575
      39892
       66836
       59838
        69980
        69811
         0169 = Basis voor diskadres.
```

De deelmethode geeft in het algemeen aanvaardbare resultaten. Er kunnen echter getallen zijn, waarbij deze methode niet zo goed voldoet. Indien we de volgende getallen delen door 10.000 zouden ze alle dezelfde rest opleveren.

38 WT 5381
39 WT 5381
43 WT 5381 namelijk de rest 5381.

Deling door een priemgetal zou in dit geval betere resultaten geven, hoewel bij deze zelfde methode andere getallen dezelfde rest kunnen opleveren.

E. Distribution method
Deze methode bestaat uit:

Het maken van een verspreidingsanalyse, om van het aantal sleutels van een hoofdbestand te bepalen in welke groep ze thuishoren. Bijvoorbeeld:

| 3 linkerposities van een artikelnummer van 7 cijfers | Aantal artikelen in deze groep |
|---|---|
| 001 - 099 | 10% |
| 100 - 350 | 35% |
| 351 - 599 | 20% |
| 600 - 799 | 25% |
| 800 - 999 | 10% |

Gebaseerd op deze analyse wordt een bepaald aantal plaatsen van het totale aantal toegekend aan iedere hoofdcategorie. Bijvoorbeeld 10% van het aantal beschikbare plaatsen wordt gereserveerd voor die artikelnummers, welke beginnen met de cijfers 001 tot en met 099.
Om de plaats van een artikel in het bestand te bepalen, wordt intern een tabel opgeslagen, welke bevat:
a. De hoogste limiet van iedere categorie.
b. Beginadres per categorie.
c. Aantal beschikbare plaatsen per categorie.

| Hoogste limiet | Beginadres | Aantal beschikbare plaatsen |
|---|---|---|
| 099 | 1000 | 100 |
| 350 | 1100 | 350 |
| 599 | 1450 | 200 |
| enz. | | |

De tabel wordt nagezocht, om van het bestand dat gebied te vinden, welke het betreffende nummer bevat.

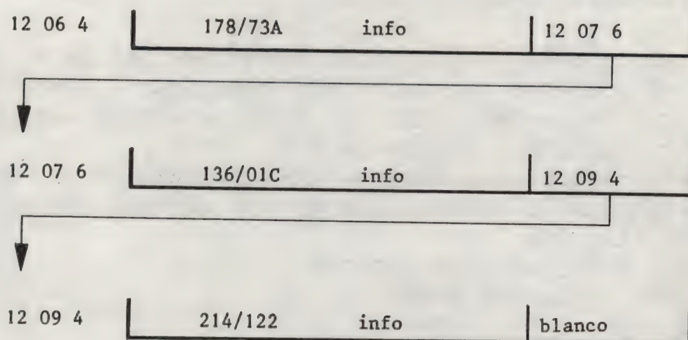
Nadat het een en ander is vastgesteld nemen we aan, dat er een komma wordt geplaatst vóór het 'Aantal beschikbare plaatsen'. De overige 4 posities van het artikel van 7 cijfers worden vermenigvuldigd met het 'Aantal beschikbare plaatsen'.
De decimalen rechts van de komma worden verwaarloosd. Dit produkt wordt opgeteld bij het 'Beginadres' om zodoende een plaats in het bestand te bepalen.
Bijvoorbeeld: Artikelnummer 0579748
057 selecteert beginadres 1000
9748 x .100 = 974
Berekend adres is 1974.

Synoniemen en overloop (overflow)
Bij vrijwel elke adresberekeningsmethode zullen meerdere sleutels hetzelfde adres opleveren. Hierdoor ontstaan synoniemen: records wier sleutel na adresberekening een adres opleveren, waar reeds een ander record is opgeborgen. Het probleem is dan een geschikte opbergplaats voor deze synoniemen te vinden (anders gezegd: "Hoe organiseer je de overflow?").
In principe zijn er twee mogelijkheden om synoniemen te relateren aan hun eigenlijke berekende adres (home-address of huisadres (vaak een spooradres) genoemd):
- Leg synoniemen vast op beschikbare ruimte zo dicht mogelijk bij het oorspronkelijk bedoelde schijfadres.
- Koppel synoniemen door middel van een ketting aan het oorspronkelijke schijfadres. Het record dat op het oorspronkelijk berekende adres opgeslagen is, krijgt dan als extra gegeven een verwijsadres naar het eerste synoniem mee. Het eerste synoniem krijgt een verwijzing mee naar het tweede synoniem etc. (een 'lege' verwijzing is dan een indicatie dat er geen verdere synoniemen zijn) (zie figuur 9).

SYNONIEMEN (in geval van indirecte adressering)

bijvoorbeeld artikelnummer 178/73A, 136/01C, 214/122 → Disk-Adres 12 06 4 (cil. Track sector)

ADRES:




Figuur 9: Verwerken van synoniemen met behulp van verwijsadressen.

De eerste methode impliceert lineair zoeken: als een record niet op het berekende adres aanwezig is, moet de omgeving nagezocht worden (bijvoorbeeld: eerst hetzelfde spoor, dan andere sporen van deze cilinder etc.).
De tweede methode impliceert het bijhouden van een wijzeradministratie voor de ketting.

Doel van het adresberekeningsalgoritme en de organisatie van de overloopruimte (opslag van synoniem) is het opereren met het bestand zo efficiënt mogelijk te maken. Dit betekent het zoeken van een compromis tussen geheugenbeslag en accesstijd (minimalisatie armbeweging bij schijven). Een belangrijk gegeven hierbij is het gebruik van een bestand: Bij een betrekkelijk statisch bestand, dat veel geraadpleegd wordt, zal het accent op 'korte zoektijd' (bijvoorbeeld kettingen) liggen. Bij een snel veranderend bestand (aanvullingen, weglating) zal het accent meer op de 'korte opbergtijd' liggen (zo goedkoop mogelijke overflow-organisatie).

Parallel aan de bovengenoemde methoden voor synoniemvastlegging zijn de volgende twee organisaties voor het overloopgebied (beide trachten armbewegingen te minimaliseren):
- Kies een zo groot mogelijk primair gebied, en een adresberekeningsalgoritme dat dit gebied 'ruim gespatieerd' opvult met records, zodat op elk spoor en op elke cilinder voldoende ruimte voor overflow is. Komt men dan synoniemen tegen dan kunnen deze in eerste instantie in hetzelfde spoor als het spoor van het huisadres geplaatst worden en vervolgens op dezelfde cilinder.
- Kies per cilinder een apart overloopgebied (analoog aan de indexsequentiële organisatie) en plaats hierin de synoniemen, waarbij alle synoniemen van een huisadres aan elkaar en het record op het huisadres gekoppeld zijn door een ketting.

Nadeel van de eerste methode is, dat synoniemen op andermans plaats terecht kunnen komen, omdat de overflowruimte van het eigen adres uitgeput is, waardoor een rommelige organisatie ontstaat.
Indien we aannemen dat de gekozen adresberekening de geconverteerde sleutels homogeen verdeelt over de beschikbare adressen (met andere woorden: gemiddeld genomen horen bij elk adres evenveel records), dan kan men met behulp van de waarschijnlijkheidsrekening de kans op synoniemen (of overflow) uitrekenen. Deze kans wordt meestal uitgedrukt als funktie van de bezettingsgraad of vullingsgraad. De bezettingsgraad is de verhouding tussen het aantal records dat in een bestand aanwezig is en het aantal records dat maximaal opgeslagen kan worden in het primaire gebied.
Bij een bezettingsgraad van 100% van het primaire gebied is de kans op synoniemen (overflowpercentage) 37% (dat wil zeggen: gemiddeld genomen zal in 37 van de 100 gevallen overflow optreden).

Bij een bezettingsgraad van 80% is het overflowpercentage 31% en bij een bezettingsgraad van 20% is het overflowpercentage nog altijd 9%.

Binning

Door een speciale techniek genaamd binning (Engels to bin = in een kist of ruimte bergen) is het mogelijk het percentage overloop te verminderen. Het beschikbare geheugen wordt dan verdeeld in stroken waarbij in iedere strook plaats is voor meerdere records. Deze stroken worden genummerd en deze nummers vormen dan de huisadressen. Dit betekent dat per huisadres plaats is voor meerdere records en per strook worden de records dan in feite sequentieel opgeborgen. Het wordt daarmee een mengsel van directe en sequentiële organisatie. Door een handige keuze van deze stroken bestaat er echter in de praktijk nauwelijks bezwaar tegen het sequentieel opzoeken. Men kiest namelijk deze stroken veelal gelijk aan een spoor, zodat huisadressen dan identiek worden met spooradressen. Aangezien men de informatie op een spoor door technische omstandigheden toch reeds sequentieel moet benaderen vormt deze methode op dit punt geen enkel bezwaar. Een dergelijke strook voor meerdere records wordt dan wel genoemd een bin (kist, blok of ruimte) of ook wel bucket.

Het effect is dat de overloop op bepaalde adressen en de tekorten op andere adressen bij grotere ruimte per adres de neiging hebben elkaar wat op te heffen. Bijvoorbeeld met maximaal 10 records per huisadres bij een bezettingsgraad van 100% van het primaire gebied daalt de overloop van 37% tot 13%, en bij 80% bezetting van 31% tot 5%.

De tabel in figuur 10 geeft de kans op overflow bij diverse 'bingrootten' en 'bezettingsgraden' van het primaire gebied. Merk op dat de overflowpercentages die hierboven genoemd zijn overeenkomen met de overflowpercentages voor bingrootte 1 (uiteraard!).

Een bezettingsgraad groter dan 1 betekent dat de overflow in een apart gebied moet worden opgeborgen omdat het primaire gebied dan te klein is om de records op te nemen. Uit de tabel is ook af te lezen dat een flinke bingrootte voordeliger is dan een groot aantal kleine bins (vergelijk bij vrijwel hetzelfde overflowpercentage van ongeveer 17.5% een bingrootte van 5 records met een bezettingsgraad van 1.0 en een bingrootte van 1 record met een bezettingsgraad van 0.4).

Een ander voorbeeld van gebruik van deze tabel is het volgende:
Laat een bestand van 4000 records opgeborgen moeten worden op een schijfsysteem met 10 sporen per cilinder en 10 records per spoor. Hoeveel cilinders zijn dan nodig als de gewenste bezettingsgraad 80% is?

Reserveren we als eerste poging per cilinder 9 primaire sporen en 1 overloopspoor dan kunnen in de primaire sporen 0.8 x 9 x 10 = 72 records een plaats vinden. Het aantal overflowrecords (als functie van de bingrootte) bij deze 72 records berekenen we als volgt:

| bin grootte | bezettingsgraad | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 | 1.1 | 1.2 |
| 1 | 4.84 | 9.37 | 13.61 | 17.58 | 21.31 | 24.80 | 28.08 | 31.17 | 34.06 | 36.79 | 39.35 | 41.77 |
| 2 | 0.60 | 2.19 | 4.49 | 7.27 | 10.36 | 13.65 | 17.03 | 20.43 | 23.79 | 27.07 | 30.24 | 33.30 |
| 3 | 0.09 | 0.63 | 1.80 | 3.61 | 5.99 | 8.82 | 11.99 | 15.37 | 18.87 | 22.40 | 25.91 | 29.33 |
| 4 | 0.02 | 0.20 | 0.79 | 1.96 | 3.76 | 6.15 | 9.05 | 12.32 | 15.86 | 19.54 | 23.25 | 26.93 |
| 5 | | 0.07 | 0.37 | 1.12 | 2.48 | 4.49 | 7.11 | 10.26 | 13.78 | 17.55 | 21.42 | 25.30 |
| 6 | | 0.02 | 0.18 | 0.67 | 1.69 | 3.38 | 5.75 | 8.75 | 12.24 | 16.06 | 20.06 | 24.11 |
| 7 | | 0.01 | 0.09 | 0.41 | 1.18 | 2.60 | 4.74 | 7.60 | 11.04 | 14.90 | 19.00 | 23.19 |
| 8 | | | 0.05 | 0.25 | 0.84 | 2.03 | 3.97 | 6.68 | 10.07 | 13.96 | 18.15 | 22.46 |
| 9 | | | 0.02 | 0.16 | 0.61 | 1.61 | 3.36 | 5.94 | 9.27 | 13.18 | 17.44 | 21.86 |
| 10 | | | 0.01 | 0.10 | 0.44 | 1.29 | 2.88 | 5.32 | 8.59 | 12.51 | 16.85 | 21.36 |
| 11 | | | 0.01 | 0.07 | 0.33 | 1.04 | 2.48 | 4.80 | 8.01 | 11.94 | 16.34 | 20.94 |
| 12 | | | | 0.04 | 0.24 | 0.85 | 2.15 | 4.36 | 7.51 | 11.44 | 15.89 | 20.58 |
| 14 | | | | 0.02 | 0.14 | 0.57 | 1.65 | 3.64 | 6.67 | 10.60 | 15.15 | 19.99 |
| 16 | | | | 0.01 | 0.08 | 0.39 | 1.28 | 3.09 | 6.00 | 9.92 | 14.56 | 19.53 |
| 18 | | | | | 0.05 | 0.28 | 1.01 | 2.65 | 5.45 | 9.36 | 14.07 | 19.16 |
| 20 | | | | | 0.03 | 0.20 | 0.81 | 2.30 | 4.99 | 8.88 | 13.66 | 18.86 |
| 25 | | | | | 0.01 | 0.09 | 0.48 | 1.65 | 4.10 | 7.95 | 12.87 | 18.31 |
| 30 | | | | | | 0.04 | 0.29 | 1.23 | 3.47 | 7.26 | 12.31 | 17.93 |
| 35 | | | | | | 0.02 | 0.18 | 0.94 | 2.98 | 6.73 | 11.86 | 17.66 |
| 40 | | | | | | 0.01 | 0.12 | 0.73 | 2.60 | 6.29 | 11.53 | 17.47 |
| 50 | | | | | | | 0.05 | 0.45 | 2.04 | 5.63 | 11.03 | 17.20 |
| 60 | | | | | | | 0.02 | 0.30 | 1.65 | 5.14 | 10.68 | 17.03 |
| 70 | | | | | | | 0.01 | 0.20 | 1.37 | 4.76 | 10.41 | 16.93 |
| 80 | | | | | | | 0.01 | 0.13 | 1.14 | 4.46 | 10.21 | 16.86 |
| 90 | | | | | | | | 0.09 | 0.97 | 4.20 | 10.05 | 16.80 |
| 100 | | | | | | | | 0.06 | 0.83 | 3.99 | 9.92 | 16.77 |

Figuur 10: Overflowpercentages bij diverse bezettingsgraden en bingrootten.

| bingrootte | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| overflow % | 20.4 | 15.4 | 12.3 | 10.3 | 8.8 | 7.6 | 6.7 |
| overflow aantal | ~ 15 | ~ 11 | ~ 9 | ~ 7 | ~ 7 | ~ 6 | ~ 5 |

We zien hieruit dat vanaf bingrootte 4 dus 9 of minder records in de overflow terecht komen, wat nog in het 10e spoor kan. Bij bingrootte 4 hebben we dan 4000/72+1 extra overflowcilinder = 57 cilinders nodig.

### 5.3.3 Reorganiseren

Ook willekeurig of direct toegankelijke bestandsorganisaties vereisen reorganisatie.
Zowel bij directe als indirecte adressering kan dit na enige tijd nodig zijn omdat de verzameling van de mogelijke sleutelwaarden veranderd is, dan wel de verdeling van feitelijke sleutelwaarden over de verzameling van mogelijke sleutelwaarden.
Veelal dient deze reorganisatie gepaard te gaan met de toepassing van een ander algoritme voor de conversie van sleutel naar adres.
Vervolgens kan bij indirecte adressering een te grote hoeveelheid synoniemen reden voor reorganisatie zijn. Afhankelijk van de organisatie van het overloopgebied kan het dan ook noodzakelijk zijn een ander algoritme voor adresberekening toe te passen. Daarnaast kan men de synoniemketens willen reorganiseren teneinde 'veel gevraagde' records voor in de ketens te plaatsen.

## 5.4 *Bestandsmutaties bij sequentiële, index-sequentiële en random bestandsorganisaties*

Een bestand is zelden een statisch geheel; gegevens worden veranderd, toegevoegd of verwijderd. De snelheid van veranderen en het belang dat men hecht aan het 'bij' zijn van het bestand bepalen de frequentie waarmee het bestand bijgewerkt moet worden.
In principe zijn er maar twee verwerkingsmethoden: sequentieel en direct.
Bij sequentiële of batchverwerking spaart men mutaties op en werkt het bestand bij met het gehele pakket.
Bij directe, in line, on line of postgewijze verwerking brengt men elke mutatie op het moment van binnenkomen in het bestand aan.
Sequentiële verwerking kan men bij elk type informatiedrager en opslagstructuur toepassen. Postgewijze verwerking laat zich praktisch niet goed toepassen bij sequentiële informatiedragers. De keuze tussen de verwerkingsmethoden wordt bepaald door de toepassingen waarvoor het bestand is opgezet. Een vliegtuigplaats reserveringssysteem zal van minuut tot minuut een bijgewerkte stand van zaken dienen weer te geven. Bij een verkoopmaatschappij moeten

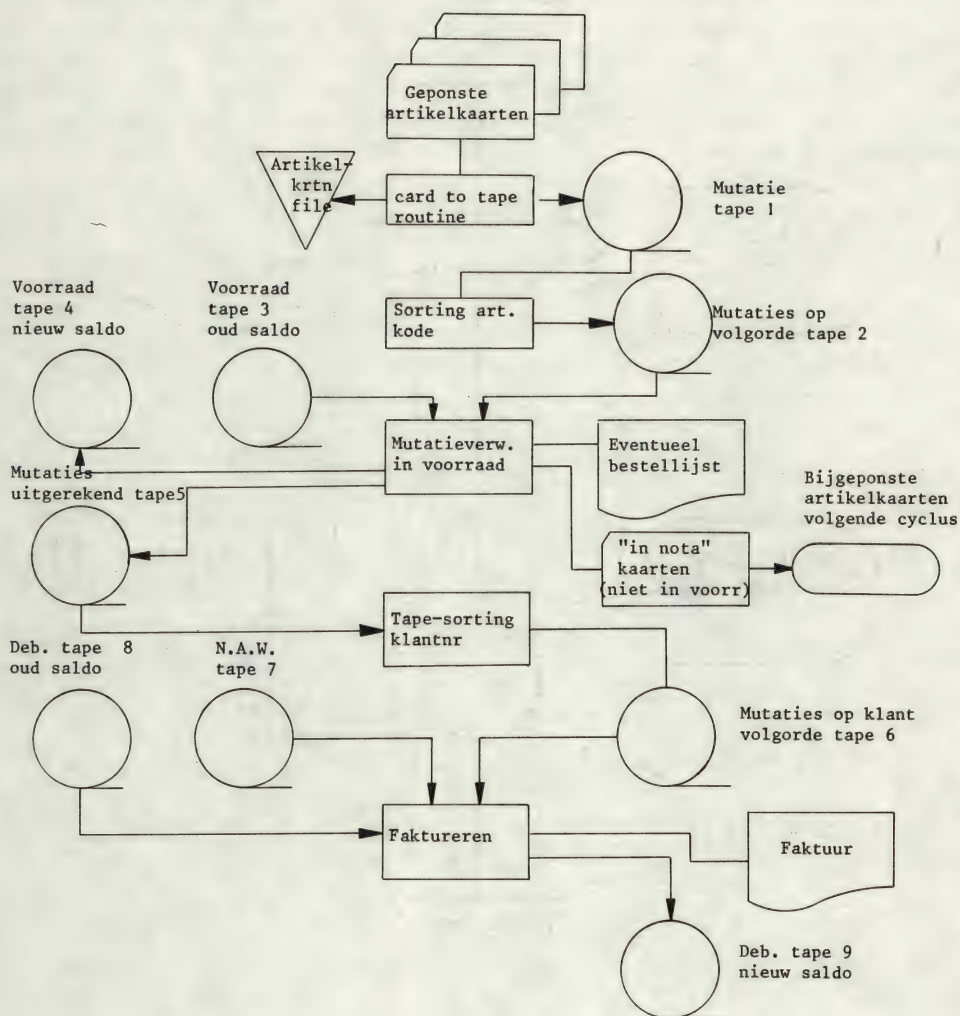
toegezegde leverenties direct van de voorraad worden afgeboekt, terwijl de fakturering ten behoeve van de uit te leveren goederen heel goed in een batch verwerkend systeem kan worden uitgevoerd. Zo ook zal een bank de actuele saldi van rekeninghouders op elk moment geregistreerd willen hebben. Als een bepaalde toepassing postgewijze verwerking nodig maakt, wil dat nog niet zeggen, dat de daartoe opgezette bestanden alleen maar postgewijs worden benaderd.
De genoemde bank zal de overschrijvingen en de produktie van dagafschriften met hetzelfde bestand realiseren, waartoe de overschrijvingen groepsgewijs worden verwerkt.
Hieronder volgt een aantal karakteristieken van batch- of groepsgewijze verwerking:

- Ieder programma is een aparte jobstep met eigen JCL ook voor de gebruikte bestanden.
- Sorteren van input en van tussenfiles is noodzakelijk.
- De te gebruiken bestanden moeten elke jobstep geopend en gesloten worden.
- Het aantal accessen is hoog vanwege tussenbestanden en omdat vaak eenzelfde record in verschillende steps wordt geraadpleegd.
- Het voordeel van batchverwerking is, dat bestanden sequentieel benaderd kunnen worden. Een masterfile is echter vaak zo groot, dat random access toch voordeliger is. Masterfiles worden vaak index-sequentieel georganiseerd. Ook 'armpje stelen' doet afbreuk aan het batchkarakter.
- 'armpje stelen'. Voor moderne toepassingen moeten veelsoortige gegevens snel en vanuit verschillende gezichtspunten toegankelijk zijn. Dit maakt soms een organisatie nodig die random access toelaat voor bepaalde toepassingen (on-line inquiries bijvoorbeeld), terwijl voor andere toepassingen sequentieel access sneller is.
  Bij sequentieel access hoeven immers bijna geen bewegingen met de lees-schrijf-koppen gemaakt te worden.
  Bij dit laatste dient voorbehoud gemaakt te worden voor het z.g. 'armpje stelen'. Indien een file een pack niet vult kan zich in een multiprogrammeringssysteem de situatie voordoen dat de sequentiële verwerking onderbroken wordt voor een access naar een ander deel van het pack (ten behoeve van een ander programma uiteraard). Men moet dan later toch met een soort direkt access terug en verliest er het voordeel van de sequentiële verwerking.
- Elke jobstep kan eenvoudig afzonderlijk worden getest. Men moet dan wel de input tussenbestanden kunnen vullen en de output tussenbestanden kunnen printen.
- De herstart mogelijkheden moeten met zorg worden nagegaan.

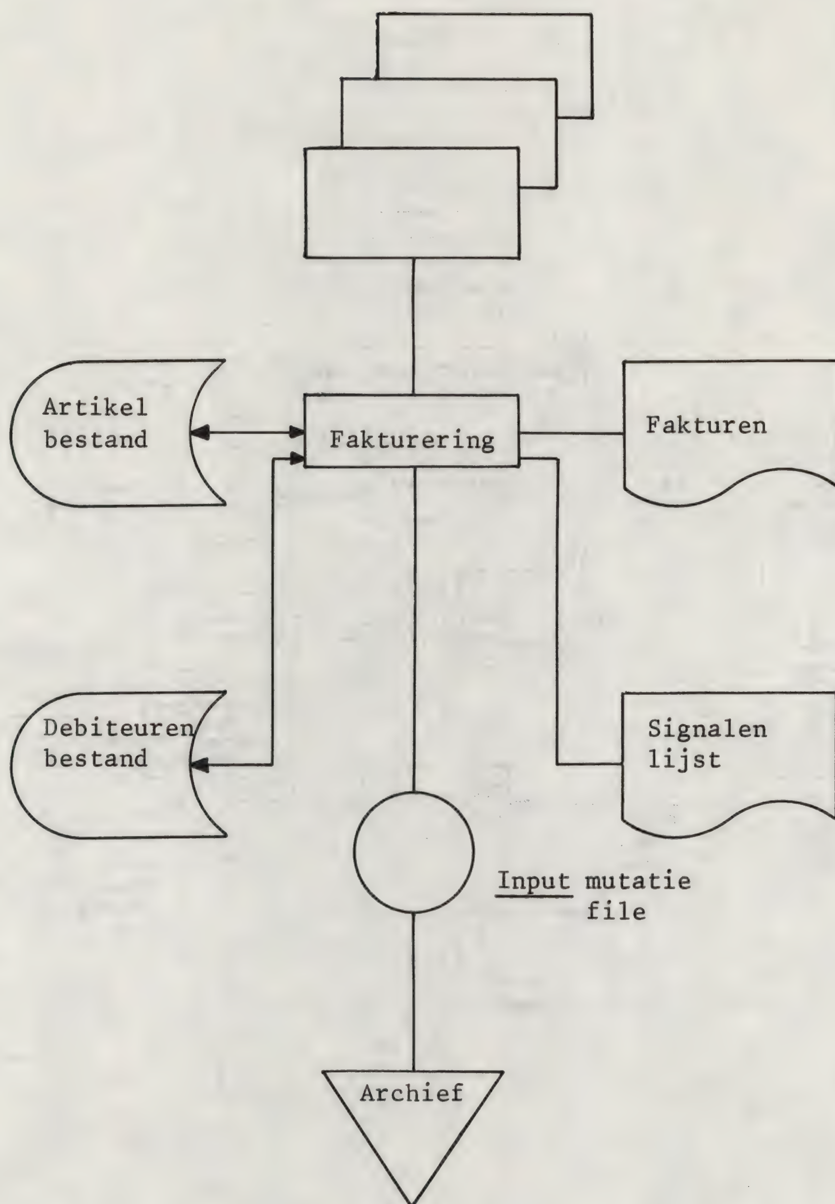
Hieronder volgen enige karakteristieken voor transactie- of postgewijze verwerking:

- De programma-opbouw volgt het verloop van de transaktie; daardoor is het systeem eenvoudiger te programmeren en een modulaire struktuur beter te verwezenlijken.
- Er zijn geen tussenbestanden nodig en ieder permanent bestand wordt maar één keer per transaktie geraadpleegd. Daardoor is er minder I/O, maar er is wel steeds random access.
- Het is niet nodig de input te sorteren. Ook geen tussentijds sorteren zoals bij batchverwerking. Dit voordeel gaat weer enigszins verloren als er periodiek een gedrukt overzicht van de transakties gemaakt wordt (bijv. voor een archief).
- Ongestoord gebruik van de partitie, doordat niet steeds jobsteps geïnitieerd en beëindigd hoeven worden.
- Restart problemen zijn betrekkelijk eenvoudig; een aantal transakties is afgewerkt, het systeem stokt op een transaktie en een aantal is niet afgewerkt.
- De overgang naar real time toepassingen is alleen mogelijk vanuit transaktiegewijze verwerking.

In de figuren 11 en 12 zijn twee systeemstroomschema's getekend voor een faktureringssysteem. Het voorbeeld van figuur 11 is uitsluitend te gebruiken voor groepsgewijze verwerking, terwijl het voorbeeld van figuur 12 geschikt is voor postgewijze verwerking. Figuur 13 geeft schematisch nog eens weer wat met post- en groepsgewijze verwerking bedoeld wordt.

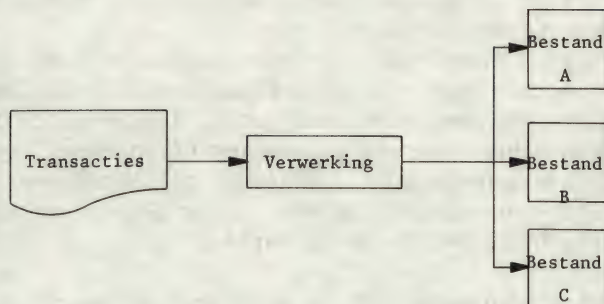



Figuur 11: Voorbeeld fakturering met behulp van een tape-systeem.

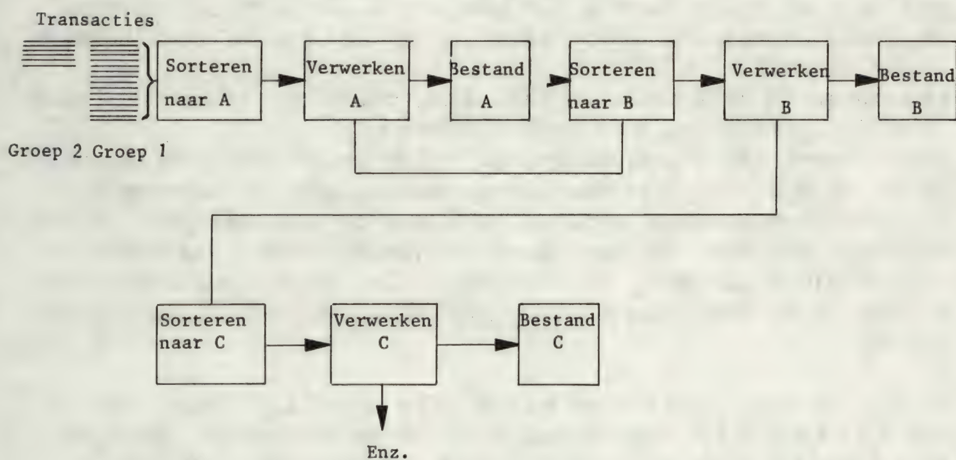



Figuur 12: Voorbeeld fakturering met behulp van een schijven-systeem.

POSTGEWIJS




GROEPSGEWIJS




Figuur 13: Voorbeeld van post- en groepsgewijze verwerking.

Is de mutatiefrequentie hoog dan zal men geneigd zijn postgewijze verwerking te kiezen. Dit vereist direct toegankelijke informatiedragers. Het probleem van het vinden van het record is eenvoudig, vergeleken bij de problemen van bestandsbijwerking en bestandsbewaking. Denkt men aan toepassing van terminals, dan kunnen van meer dan een kant mutaties voor hetzelfde record binnenkomen. Men ontmoet hier dan ook problemen van deadlock, resource sharing, die bij bedrijfssystemen behandeld zijn.
Voorts heeft men bij direct toegankelijke bestanden - als men geen voorzorgsmaatregelen treft - slechts één versie van het bestand, terwijl bij sequentiële bestanden men door het type informatiedrager noodgedwongen beschikt over enige versies (grootvader, vader, zoon-systeem bij magneetbanden). Waar mogelijk zal men bij online-bestanden on-line-vragen toelaten, maar pogen om mutaties op te sparen en deze batchgewijs te verwerken.

Bestand met sequentiële opslagstructuur
De verwerkingswijze is veelal sequentieel. De batchmutaties worden op hetzelfde criterium (bijvoorbeeld de sleutel) gesorteerd als het bestand.
Als het bestand op magneetband staat, is verwerking in principe simpel: op grond van sleutelvergelijking worden beurtelings wagens (records) uit 'trein records oud bestand' en uit de trein 'mutaties' afgehaakt en na mutatieverwerking aangehaakt aan de trein 'records nieuw bestand'.
Opschuifproblemen doen zich niet voor, omdat het hele bestand al of niet gemuteerd, gecopieerd wordt.
Bij een sequentieel bestand op schijfgeheugen doet zich het opschuifprobleem wel voor. Veelal zal men 'weglatingen' niet uitvoeren, maar het desbetreffende record markeren en 'toevoegingen' via verwijzingen koppelen aan hun eigenlijke fysieke plaats. Periodieke reorganisatie is dan nodig om weer een zuiver sequentieel bestand te krijgen. Deze reorganisatie geschiedt weer wel met behulp van copiëren.

Bestand met index-sequentiële opslagstructuur
Het veranderen van records in een index-sequentieel georganiseerd bestand is op het eerste gezicht een eenvoudige bewerking: een record kan direct opgespoord worden met de sleutel van het transactierecord, veranderd en dan weer teruggeschreven. Omdat mogelijk meerdere mutaties op een oud record moeten worden aangebracht is het bij een hoge file volatility efficiënter om de transactierecords toch te sorteren op sleutelvolgorde.

Het toevoegen van records in een index-sequentieel bestand is daarentegen niet een simpele bewerking omdat veelal de indextabel (len) bijgewerkt moet(en) worden. Vooral wanneer de toe te voegen

records in het overloopgebied terecht komen, moet enige administratie in de tabellen en/of in kettingwijzers in de records in het overloopgebied uitgevoerd worden. Als het werkgeheugen te klein is om complete sporen van een willekeurig toegankelijk geheugensysteem op te nemen, kunnen ingewikkelde programma's hiervan het gevolg zijn.

Ook het verwijderen van records kan aanleiding geven tot het bijhouden van indextabel(len), maar vaak kiest men de oplossing dat een overbodig geworden record als zodanig 'gemerkt', doch niet verwijderd wordt.
Wanneer na enige tijd grote aantallen records in overloopgebieden terecht zijn gekomen, en/of overbodig geworden records flink in aantal zijn toegenomen, dan wel een overloopgebied vol is geraakt, moet men met het oog op toegenomen verwerkingstijden overgaan tot een bestandsreorganisatie. Dit gebeurt door het totale bestand in sleutelvolgorde te copiëren op een nieuwe schijveneenheid, en daarbij overbodige records te verwijderen en overloopgebieden leeg te maken. Bij deze verwerking zal veelal tevens een copie van het bestand op magneetband vastgelegd worden ten behoeve van bestandsbeveiliging.

Bestand met willekeurig toegankelijke opslagstructuur
Bij een directe relatie tussen recordsleutel en recordadres is de techniek voor het vinden van records triviaal; het record is in principe in één accesstijd beschikbaar.
Bij een tabelrelatie is de zoektechniek uiteraard dat in de tabel (en eventueel de subtabellen) naar de recordsleutel gezocht wordt, waarna het recordadres bekend is. De zoektijd in de tabellen is gemiddeld van de in de ordegrootte van microseconden als de tabel voor n records al in het kernengeheugen aanwezig is, maar van de orde van de accesstijd (milliseconden) als de tabel nog op een secundair geheugensysteem staat. Om het record zelf in het kernengeheugen te krijgen moet één accesstijd opgeteld worden. De zoektijd in de tabel, hoewel reeds kort, kan nog wat verbeterd worden door de recordsleutels in de tabel te sorteren en dan de binaire zoekmethode te gebruiken. (Tegenover een korte zoektijd staat bij de gesorteerde tabel uiteraard een langere opbouwtijd van de tabel omdat daarin de sleutels gesorteerd moeten worden. Het aantal keren dat de tabel opgebouwd moet worden versus het aantal keren dat de tabel voor een zoekproces gebruikt moet worden, bepaalt dan de keus tussen gesorteerde en ongesorteerde tabellen.)

Bij een functionele relatie tussen recordsleutel en recordadres moet voor het zoeken van een record uiteraard eerst een sleutelconversie worden uitgevoerd met de sleutel als gegeven. Bij een ge-

schikte 'randomizing' methode zal slechts nu en dan een synoniem optreden, zodat aangezien de berekeningstijd meestal te verwaarlozen is, gemiddeld in (iets meer dan) één accesstijd een record beschikbaar zal zijn. Is het aantal synoniemen daarentegen niet te verwaarlozen, dan kan voor het zoeken eventueel een groot aantal accessen nodig zijn.

Samenvattend kan gezegd worden dat het probleem meestal zit in het vinden van een goede 'randomizer', doch dat daarna niet alleen het zoeken, maar ook het toevoegen, veranderen of verwijderen van records weinig problemen geeft en snel is. Omdat iedere zoek ook onafhankelijk van de vorige is, hoeven ook de transacties niet gesorteerd aangebracht te worden. Deze bestandsstructuur is dan ook zeer geschikt voor 'on-line' toepassingen, al zullen zoals reeds eerder gezegd door het systeem voorzieningen getroffen moeten worden om 'bijna gelijktijdige' mutaties op een bepaald record goed te verwerken.

## 5.5 *Andere bestandsorganisatievormen*

### 5.5.1 Relatieve bestandsorganisatie

Bij deze bestandsorganisatie is er een relatie tussen de sleutel en de relatieve positie van het record ten opzichte van het begin van de file. Deze relatie kan simpel zijn: 'de sleutel is het rangnummer van het record in de file', maar kan ook via een tabel of algoritme vastgelegd zijn.

### 5.5.2 Inverted file structuren (geïnverteerd bestand)

De tot dusver besproken structuren zijn min of meer geschikt voor 'searching', dus voor het vinden van gegevens op grond van de sleutel van een record. Wanneer het echter om een 'retrieval' probleem gaat, dat wil zeggen het vinden van records waarvan bepaalde items aan zekere voorwaarden (meestal gelijkheid of ongelijkheid) moeten voldoen, zit er met de behandelde structuren vaak niet veel anders op dan alle records van een bestand te lezen, de items waar het om gaat te onderzoeken en de records die voldoen aan de voorwaarden weg te schrijven op een andere informatiedrager. Het volledige bestand moet zo doorgewerkt worden, hetgeen voor zeer grote bestanden kostbaar kan zijn.

Wanneer het aantal voor een retrieval relevante items, vermenigvuldigd met het aantal mogelijke waarden dat ieder van die items kan aannemen, niet al te groot en het oorspronkelijke grote bestand willekeurig toegankelijk is, kan een inverted file goede uitkomst

bieden. In feite is deze te vergelijken met een trefwoordensysteem van de systematische catalogus van een grote bibliotheek. Men bouwt namelijk bij het oorspronkelijke bestand een geordend hulpbestand - de 'inverted file' - op, waarvan de recordsleutels bepaald worden door de namen en mogelijke waarden van voor retrieval relevante items. De items van de inverted file records zijn dan de (gesorteerde) sleutels (of fysieke adressen) van alle records van de oorspronkelijke file, waarvan de corresponderende items aan dezelfde bepaalde voorwaarde voldoen. Een retrievalvraag laat zich met behulp van logische a n d, o r en n o t operatoren formuleren; bijvoorbeeld een retrieval van die records in een bevolkingsadministratie behorende bij bewoners die èn getrouwd zijn èn drie minderjarige òf geen kinderen hebben, maar niet een vrij beroep hebben (is deze zin eigenlijk zonder gebruik van haakjes wel éénduidig te interpreteren?). Met behulp van de inverted file vindt men dan direct de recordsleutels (of recordadressen) van de gewenste records in het oorspronkelijke bestand, immers in het hulpbestandstuk 'gehuwd' vindt men de recordsleutels van alle gehuwde bewoners, in het hulpbestandstuk 'drie minderjarige kinderen' de recordsleutels van alle bewoners met drie minderjarige kinderen, in het hulpbestandstuk 'nul kinderen' die van alle bewoners met nul kinderen en in het hulpbestandstuk 'vrij beroep' tenslotte de recordsleutels van alle bewoners met een vrij beroep. Door dan overeenkomstig de retrievalvraag logische bewerkingen op de groepen recordsleutels uit te voeren isoleert men tenslotte de recordsleutels van de gewenste records, die dan zo mogelijk direct opgezocht kunnen worden. De groepen recordsleutels moeten voor een efficiënte logische bewerking uiteraard gesorteerd zijn.

Geïnverteerde bestanden worden veelal op de volgende twee wijzen geïmplementeerd:
- Ten eerste door inderdaad afzonderlijke hulpbestanden waarbij dan eigenlijk gewerkt wordt met variabele recordlengte, want ieder record bevat naast de itemwaarde een variabel aantal adressen (van records die bij deze waarde van dit item horen).
- De tweede methode is met kettingadressen in het hoofdbestand. Voor iedere gewenste itemwaarde is in ieder record dan een kettingadres aanwezig. Verder wordt voor iedere itemwaarde een afzonderlijke index bijgehouden die het eerste adres van de betrokken ketting geeft. O o k h i e r g e l d t w e e r d a t d e k e t t i n g e n n i e t t e l a n g e n h e t a a n t a l k e t t i n g e n n i e t a l t e g r o o t m o g e n w o r d e n. Een verdere consequentie is dat bij iedere mutatie zonodig ook de kettingen weer moeten worden bijgewerkt. Bijvoorbeeld in het gegeven voorbeeld van woonplaatsen moet iemand na verhuizing naar een andere stad in een andere ketting worden gehangen. Zou men in dit voorbeeld het probleem hebben dat relatief veel mensen in bepaalde plaatsen wonen dan zouden daardoor de ket-

tingen lang worden. Men moet voor die gevallen dan de index verfijnen door bijvoorbeeld die woonplaatsen op te delen in wijken en deze wijken dan als zelfstandige woonplaatsen te beschouwen. Hoe men dit precies moet doen is in het algemeen niet aan te geven en hangt volkomen af van de onderhavige toepassing.

### 5.5.3 Keten of kettingadressering

Lijststructuren, gerealiseerd door middel van kettingen, kunnen zoals we gezien hebben een bepaalde samenhang (bijvoorbeeld een sequentiële) representeren, zonder dat deze samenhang door een fysieke ligging vastgelegd hoeft te worden (geen opschuifprobleem, wel periodiek reorganiseren om genoeg vrije ruimte te krijgen). Vaak zal men naast een keten van records ook een keten van vrije plaatsen bijhouden.

Voorbeeld:

| adres | sleutel | informatie | kettingadres |
|---|---|---|---|
| 1 | 2355 | - | 10 |
| 2 | 488 | - | 11 |
| 3 | vrij | - | * |
| 4 | 1012 | - | 1 |
| 5 | 365 | - | 2 |
| 6 | vrij | - | 8 |
| 7 | 743 | - | 4 |
| 8 | vrij | - | 3 |
| 9 | vrij | - | 6 |
| 10 | 4768 | - | * |
| 11 | 513 | - | 7 |

begin recordketting: 5      *: einde ketting
begin vrije ketting : 9
Ketting op volgorde van opklimmende sleutel.

Na toevoeging van record met sleutel 3533 en verwijdering van record 488 ziet de ketting er als volgt uit:

| adres | sleutel | informatie | kettingadres |
|---|---|---|---|
| 1 | 2355 | - | 9 |
| 2 | vrij | - | * |
| 3 | vrij | - | 2 |
| 4 | 1012 | - | 1 |
| 5 | 365 | - | 11 |
| 6 | vrij | - | 8 |
| 7 | 743 | - | 4 |
| 8 | vrij | - | 3 |

| | | | |
|---|---|---|---|
| 9 | 3533 | - | 10 |
| 10 | 4768 | - | * |
| 11 | 513 | - | 7 |

Opgave: Ga na of dit juist is.

Nadelen van de kettingadressering zijn:
- Wijzers moeten aan de recordinformatie toegevoegd worden.
- Administratie vrije ruimte moet bijgehouden worden.
- Records komen kris kras door het geheugen te liggen. Bij semi-direct toegankelijke geheugens als schijvengeheugens wordt daardoor het aflopen van lange kettingen inefficiënt (kans op wachten op omwenteling of armbeweging is groot).

Men kan met name het laatste nadeel opvangen door lange kettingen niet als een sequentiële rij maar als boom met meerdere niveaus op te zetten (zie figuur 14).

Kettingadressering op beperkte schaal wordt ook bij de andere genoemde bestandsorganisaties gebruikt (bijvoorbeeld om synoniemen te ketenen).

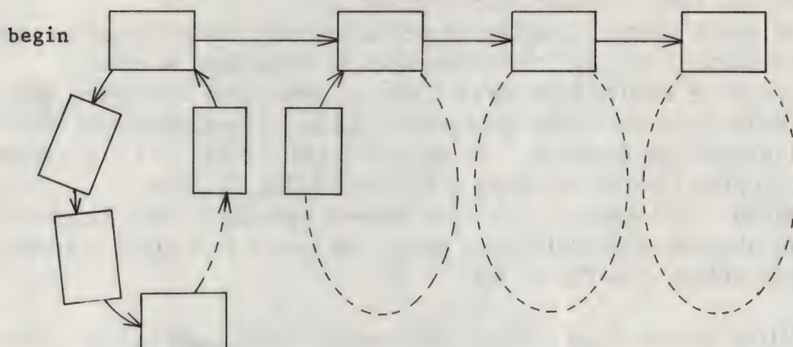
In het navolgende worden onder A tot en met D enige voorbeelden van de toepassing van kettingadressering behandeld.

A. Produktstructuurbestanden (Voorbeeld van een relatie tussen twee bestanden)

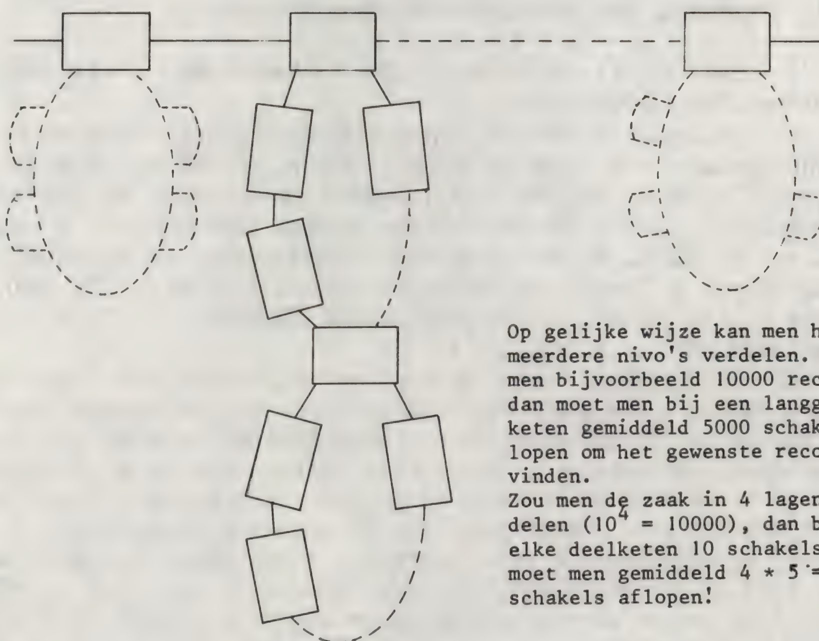
Deze worden onder andere toegepast in fabrieksadministraties van assemblagebedrijven. Hierbij wordt van elke produktsamenstelling vastgelegd uit welke subsamenstellingen en onderdelen een produkt is opgebouwd. Deze structuur is schematisch weergegeven in figuur 15. In dit voorbeeld is het eindprodukt A opgebouwd uit de subsamenstellingen E, H en K en uit de onderdelen 1, 4 en 6. Het eindprodukt B is opgebouwd uit de subsamenstellingen E, K, IJ en Z en uit de onderdelen 1, 4, 6, 7, 8 en 12.
Elke subsamenstelling vormt nu een 'list' en wordt tijdens de verwerking vaak als één geheel behandeld. Voor een nadere beschouwing van de werkzaamheden die met deze gestructureerde informatiebestanden moeten worden uitgevoerd beschouwen we nu uitsluitend de methode die betrekking heeft op de onderdelenreserveringen en het aanbrengen van wijzigingen in de produktiesamenstelling.
In figuur 16 zijn hiervoor twee bestanden schematisch weergegeven. Ten eerste het hoofdbestand, dat onder andere de voorraad van alle onderdelen, materialen en subsamenstellingen weergeeft en waarbij voor identificatie van elk element in het bestand alleen het artikelnummer is gebruikt. Ten tweede een produktstructuurbestand waarin voor elke samenstelling en subsamenstelling een gegeven is opgenomen, dat geïdentificeerd wordt met een samenstellingsnummer en waarin is vastgelegd uit welke onderdelen en subsamenstellingen




In twee lagen zou het er als volgt uit zien




met 3 lagen schematisch

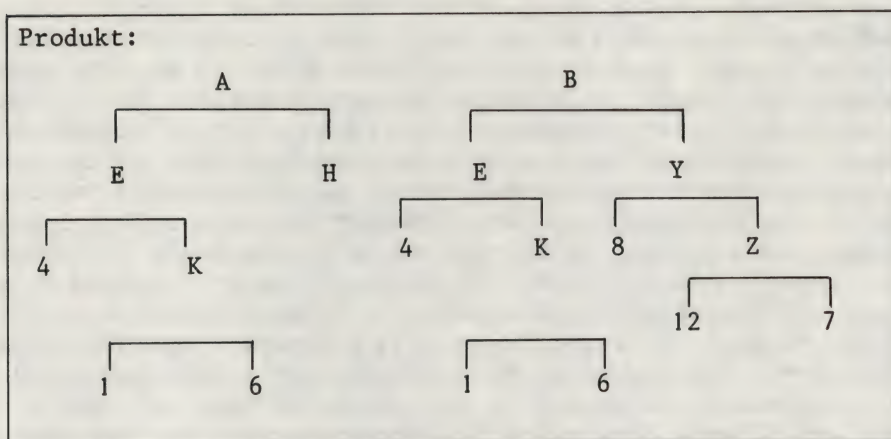


Op gelijke wijze kan men het in meerdere nivo's verdelen. Heeft men bijvoorbeeld 10000 records dan moet men bij een langgerekte keten gemiddeld 5000 schakels aflopen om het gewenste record te vinden.
Zou men de zaak in 4 lagen verdelen ($10^4 = 10000$), dan bevat elke deelketen 10 schakels en moet men gemiddeld $4 * 5 = 20$ schakels aflopen!

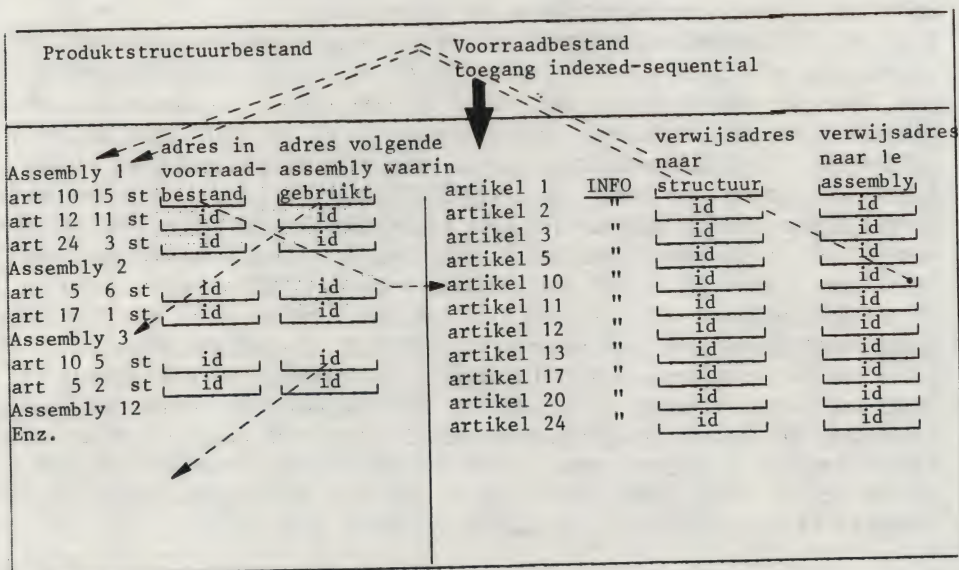
Figuur 14.

het betrokken produkt is opgebouwd. Wanneer een werkorder voor 100 stuks van produktsamenstelling 1 moet worden verwerkt, dan kan de hiervoor noodzakelijke materiaalreservering als volgt plaatsvinden: Met behulp van de indexed-sequential methode wordt in het voorraadbestand de vastgelegde informatie van artikel 1 opgezocht. In dit record bevindt zich naast de normale informatie ook een verwijsadres naar het bijbehorende record in het structuurbestand. In dit record is dus de produktsamenstelling van samenstelling 1 opgenomen. De computer kan nu bijvoorbeeld berekenen dat 1500 stuks van onderdeel 10 nodig zijn. Ook kan een verdere verwerking in het voorraadbestand plaatsvinden door in deze vastgelegde samenstellingsinformatie een verwijsadres op te nemen naar onderdeel 10 in het voorraadbestand. Voor elk voorraadrecord en elke samenstellingsinformatie kan deze werkwijze worden gevolgd. De indexed-sequential toegangsmethode tot het hoofdbestand is hier als uitgangspunt gekozen omdat voor het vervaardigen van de bijbehorende overzichten onder meer de alfabetische informatie zoals artikelnaam nodig is. Terwille van een compacte vastlegging is deze informatie uitsluitend in dit voorraadbestand opgenomen. Het vastleggen van deze informatie in het produktstructuurbestand zou veel herhalingen van informatie tot gevolg hebben, daar bepaalde onderdelen in verschillende produktsamenstellingen voorkomen.
Een tweede bewerking die met deze beide bestanden uitgevoerd moet worden is het vastleggen van wijzigingen in de produktsamenstellingen. Stel dat bijvoorbeeld onderdeel 10 niet blijkt te voldoen en dat hiervoor een ander onderdeel is ontworpen dat codenummer 35 heeft gekregen. Voor het aanbrengen van deze wijzigingen zal men moeten beschikken over een toegangsmethode tot alle records in het produktstructuurbestand waarin onderdeel 10 is opgenomen. Hiervoor wordt eerst het voorraadbestand met behulp van de indexed-sequential methode geraadpleegd en het record van onderdeel 10 bereikt. Daar in dit record ook het adres opgenomen is dat verwijst naar het produktstructuurbestand, en wel naar het eerst c.q. laagst genummerde samenstellingsadres waarin dit artikel is opgenomen, kan de wijziging ook hierin worden aangebracht. Tevens komt het verwijsadres beschikbaar dat het volgende samenstellingsadres aangeeft waar ditzelfde artikel is opgeborgen. Deze verwijsadresprocedure zet zich voort, totdat het laatste samenstellingsadres is bereikt waarin onderdeel 10 is opgenomen (zie pijlen in figuur 16).

Deze methode maakt het mogelijk zeer snel ingewikkelde procedures uit te voeren. Bij het gebruik van magnetische bandeenheden of een andere bestandsorganisatiemethode zijn voor het uitvoeren van dezelfde procedures meer bewerkingsfasen nodig dan bij gebruik van schijvengeheugens en deze 'list-processing' methode het geval is.




Figuur 15: Structuur produkt samenstelling.




Figuur 16: Structuur bestanden.

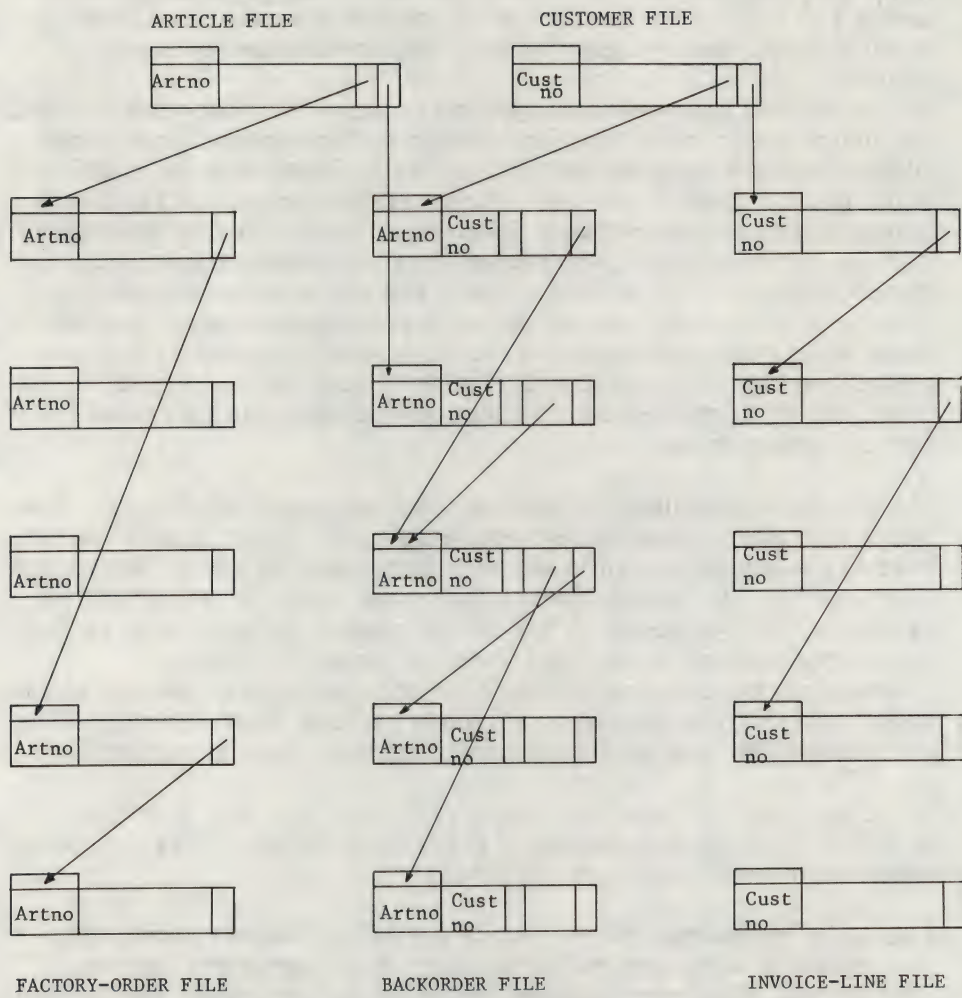
B. Relatie tussen bestanden
Kettingadressering is niet alleen zeer nuttig om records binnen een bestand in volgorde te houden, maar evenzo om records van een bestand te verbinden met gerelateerde informatie van een ander bestand.
Een voorbeeld hiervan wordt getoond in figuur 17. Elk artikelrecord kan een of meer fabrieksorders hebben die op verschillende tijden tijdens verwerking ontstaan zijn. Het artikelrecord in het hoofdbestand is door middel van een 'pointer' verbonden met de eerste fabrieksorder voor dat artikel. Wanneer er meerdere fabrieksorders zijn dan verwijst het eerste record naar het tweede enzovoorts. Op dezelfde manier zijn de faktuurlijnen van een klant verbonden met klantenrecords. Het bestand van wachtende klantenorders (backorders) is iets meer gecompliceerd, want iedere backorder behoort niet alleen bij een bepaald artikel maar evenzo bij een bepaalde klant. Er zijn dus theoretisch evenveel kettingen als er artikelrecords tezamen zijn.

In figuur 18 liggen deze relaties nog iets gecompliceerder want hier wordt niet alleen gewerkt met een ketting, die iedere keer verwijst naar het volgende record in de keten beginnend bij het eerste record ('forward' of 'downward' chain), maar ook wordt er een ketting bijgehouden, die beginnend bij het laatste record terugverwijst tot het eerste record bereikt is ('backward' of 'upward' chain).
Wanneer uit het backorderbestand records verwijderd moeten worden zullen ook alle voorgaande en volgende records in de diverse ketens gelezen moeten worden om de verwijzingen correct te houden.

C. Toevoegen aan en verwijderen uit de ketting
In figuur 19 wordt een voorbeeld gegeven op welke manier een record toegevoegd moet worden in een keten.

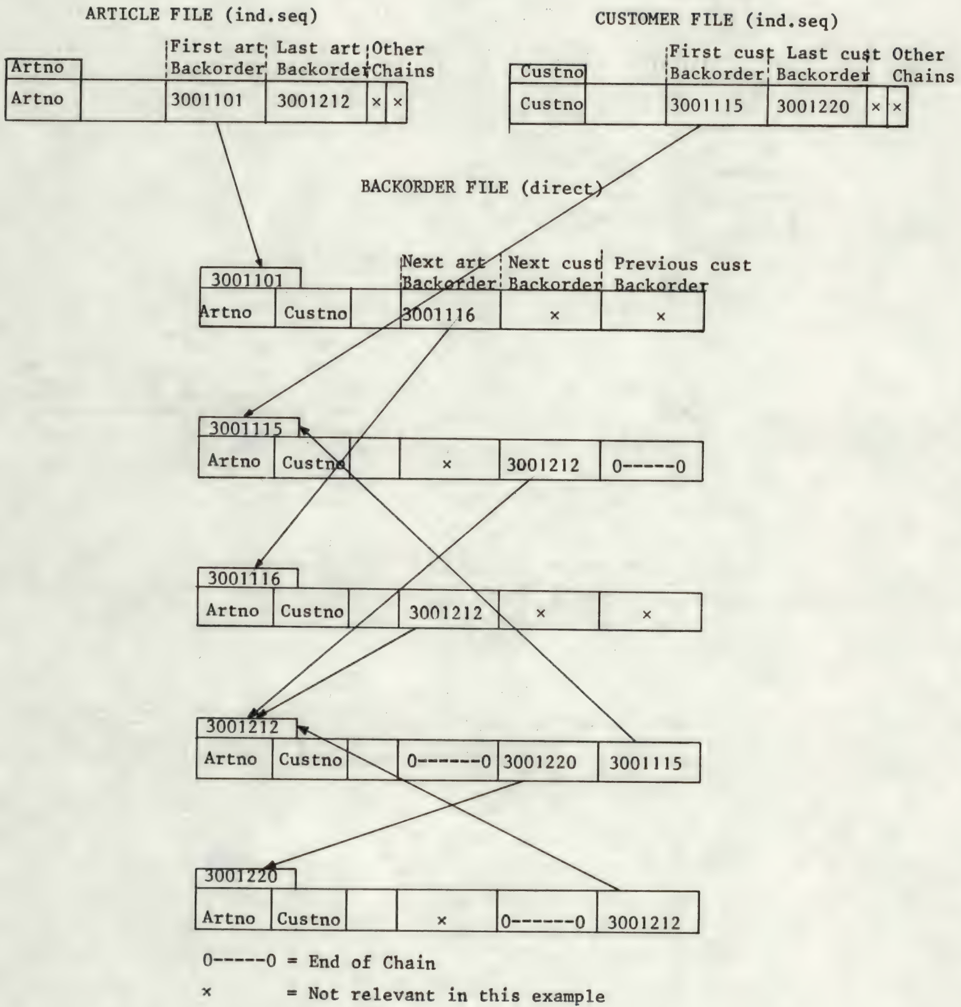
1 a. Lees het laatste record van de artikel-backorder keten. Het absolute adres kan in het 'masterrecord' gevonden worden.
  b. Verander het verwijsadres (nu nullen hetgeen einde van de keten indiceert) in het absolute adres dat in het geheugen gevonden kan worden als eerste vrije record in het bestand (NFR = Next Free Record).
  c. Schrijf de laatste 'backorder' terug.
  d. Verander het verwijsadres naar de laatste 'backorder' in het 'masterrecord' in het adres dat in NFR gevonden is.

2 Dezelfde procedure moet gevolgd worden voor de klanten 'backorder' keten. Omdat een terugwaartse keten nodig is, moet de vroegere inhoud van het verwijsadres naar de laatste backorder bewaard blijven in een werkveld (hier genoemd PCR = Previous Customer Record).

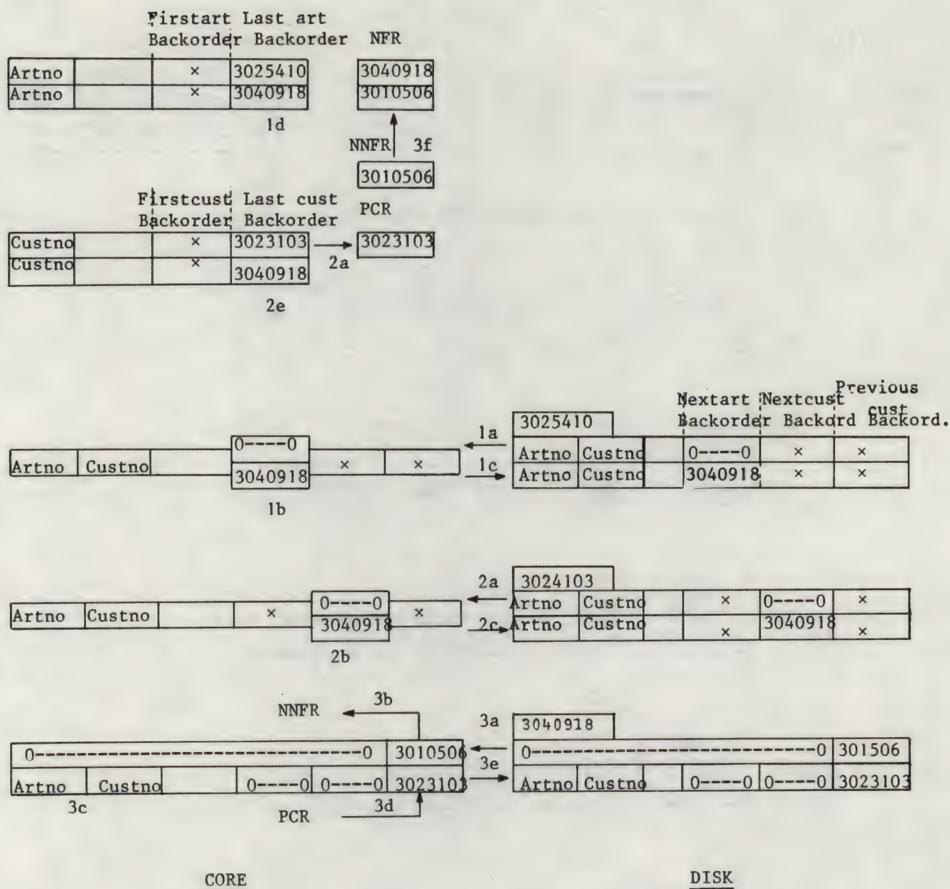



Figuur 17: Relatie tussen bestanden.

Figuur 17: Relatie tussen bestanden.




Figuur 18: Gebruik van kettingen.




Figuur 19: Toevoegen aan de ketting.

3 a. Lees het vrije record, waarvan het adres in NFR staat.
b. Bewaar het adres naar het volgende vrije record in een werkveld (NNFR = Next Next Free Record).
c. Creëer de nieuwe backorder in het outputgebied in het geheugen.
d. Breng het adres opgeborgen in PCR naar het veld 'voorgaande klanten backorder' in het nieuwe backorderrecord.
e. Schrijf het geformeerde backorderrecord terug op de plaats, waarvan het adres in NFR staat.
f. Breng het absolute adres van het volgende vrije record van NNFR naar NFR.

4 Trek één af van de inhoud van een veld in het geheugen dat het aantal vrije records aangeeft (AFR = Available number of Free Records).

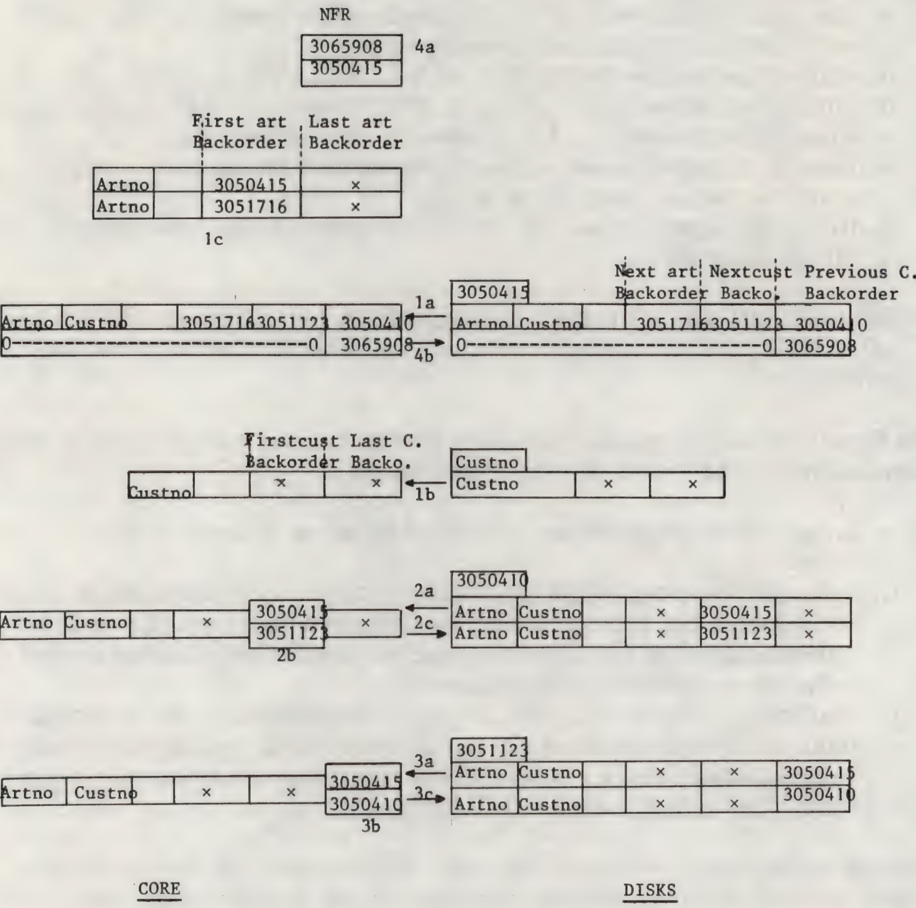
In figuur 20 wordt een voorbeeld gegeven hoe een record uit het midden van een keten moet worden verwijderd.

1 a. Lees het te verwijderen record; het adres staat in het masterrecord.
b. Lees het overeenkomstige klantenrecord, de symbolische sleutel staat in het juist gelezen backorderrecord (1c). Deze symbolische sleutel kan gebruikt worden omdat het klantenbestand indexed-sequential georganiseerd is.
c. Verander het adres in het artikel masterrecord, dat verwijst naar het te verwijderen backorderrecord, in het adres van de volgende backorder, welke in het juist gelezen backorderrecord staat (1a) onder 'next article backorder'.

Om de ketens (zowel 'down' als 'up') gerelateerd aan het klantenrecord te herstellen moeten de volgende acties ondernomen worden:

2 a. Lees de voorgaande klantenbackorder, het adres hiervan wordt gevonden in de zojuist gelezen backorder onder 'previous customer backorder'.
b. Verander het verwijsadres in deze backorder (2a), die nu nog verwijst naar de te verwijderen backorder, in het adres van de nu volgende backorder. Het adres hiervan staat in de te verwijderen backorder onder 'next customer backorder'.
c. Schrijf de backorder, die gelezen is in 2a, terug.

3 a. Lees met het adres dat staat in 'next customer record' in het te verwijderen backorderrecord, de volgende backorder.
b. Verander het adres in dit backorderrecord dat verwijst naar de voorgaande, in het adres dat in de te verwijderen backorder staat als 'previous customer record'.
c. Schrijf de backorder, gelezen in 3a, terug.

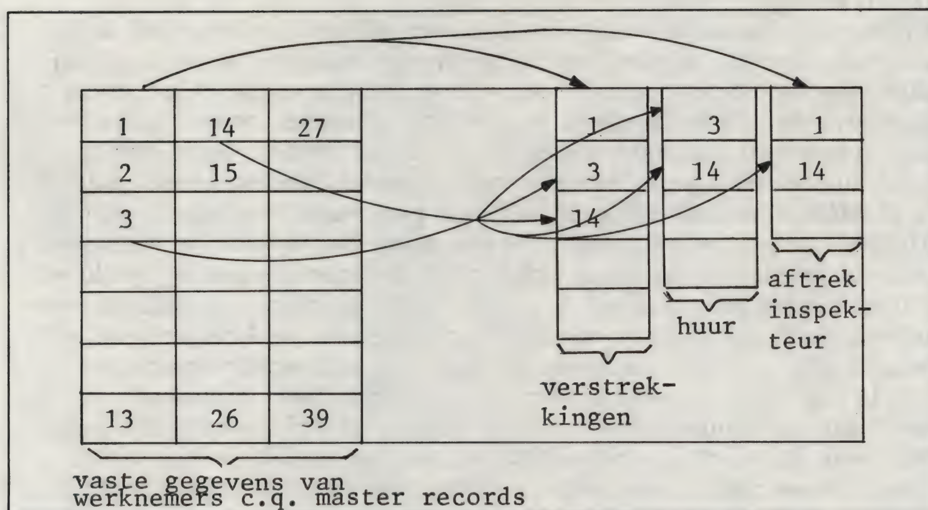



Figuur 20: Verwijderen uit een ketting.

Figuur 20: Verwijderen uit een ketting.

### D. Gesegmenteerde records

Dit soort informatie-eenheden die men ook wel 'master and detail records' noemt, worden vooral toegepast wanneer in een bestand niet alleen de lengte van de records onderling variabel is, maar ook de hoeveelheid informatie of het aantal rubrieken per records steeds aan veranderingen onderhevig is. Voor het vervaardigen van de loonslips kunnen we in een personeelsbestand bijvoorbeeld rubrieken aantreffen voor inhoudingen op grond van: verstrekkingen van werkkleding, schoeisel en dergelijke die gedeeltelijk door de werknemer betaald worden in de vorm van periodieke inhoudingen op het loon; aftrek inspecteur wanneer belastingvermindering wegens bijzondere lasten wordt toegestaan; woninghuur indien de werkgever beschikt over eigen woningen en deze verhuurt aan zijn werknemers.
Het is duidelijk dat slechts een beperkt aantal werknemersrecords een rubriek woninghuur en aftrek inspecteur zal hebben. Dit geldt ook voor de rubriek verstrekkingen, echter met dit verschil dat vooral deze rubriek een hoge omloopsnelheid heeft. Wanneer het totaal in te houden bedrag ƒ 125,- bedraagt, zal na 5 wekelijkse inhoudingen van elk ƒ 25,- deze rubriek voor de betrokken werknemer voor onbepaalde tijd niet meer nodig zijn. Verder zijn deze inhoudingen voor verstrekkingen, hoewel van toepassing op bijna alle werknemers ook nog over lange periode zeer verspreid in het bestand opgenomen. Werkkleding zal niet aan alle werknemers tegelijkertijd en op vooraf vastgestelde data worden uitgereikt. Dit is slechts een zeer beknopt voorbeeld. In de praktijk leidt dit soort situaties vaak tot veel grotere variaties in lengte binnen één record dan hier tot uitdrukking komt. Voor bestanden die bovengenoemde karakteristieken hebben, biedt het gebruik van gesegmenteerde records een goede oplossing. Bij deze methode wordt het bestand gesplitst in verscheidene bestanden, waarbij elk bestand slechts een bepaald deel van de informatie per onderwerp bevat. Met behulp van verwijsadressen kunnen dan de diverse informatiesegmenten aan elkaar 'gekoppeld' worden voor verwerking. Elk volledig record bestaat dus uit één of meer segmenten. Op deze wijze zijn records verkregen met een variabele informatielengte. Daar bij dit soort bestanden de informatie steeds in beweging is, zijn de procedures voor het bijwerken van het gehele bestand omvangrijker. Ook zal de frequentie waarmee deze procedures moeten worden uitgevoerd hoger zijn dan bij andere recordtypen het geval is. Hier staat echter tegenover dat met deze aanpak een veel efficiënter gebruik wordt gemaakt van de externe geheugencapaciteit.
Zie voor de schematische voorstelling van gesegmenteerde records figuur 21. De pijlen geven de functie van de in de 'vaste-man' records opgenomen verwijsadressen weer.




Figuur 21.

## 5.6 *Vergelijking van de sequentiële, index-sequentiële en direct toegankelijke organisatie*

Van de drie beschreven bestandsorganisatiemethoden, de sequential organisatie, de indexed-sequential organisatie en de random organisatie wordt bij de laatste twee methoden volledig gebruik gemaakt van de specifieke mogelijkheden die direct toegankelijke externe geheugens bieden. De keuze van de methode moet altijd gemaakt worden op basis van de eisen voor alle bewerkingen waarbij het bestand betrokken is. Er zijn echter nog andere factoren, die deze keuze beïnvloeden. Hiertoe behoren de beschikbare computerconfiguratie, de mate waarin een bestand statisch of dynamisch is, en de omvang van het bestand ten opzichte van de externe geheugencapaciteit en de aanwezigheid van standaardprogrammatuur.
De realisatie van programmatuur op het gebied van bestandsorganisatie is een zeer gecompliceerde zaak. Het betreft namelijk niet alleen het samenstellen van een programma voor een bestandsorganisatie. Een dergelijk programma moet meestal ook werken in een systeem van multiprogrammering, multiverwerking of tijdscharing. Daarbij komen problemen als beveiliging van bestanden toegestaan en verboden wederzijds gebruik van bestanden door verschillende programma's aan de orde. Het gevolg is dat die programma's niet meer als afzonderlijke applicatieprogramma's kunnen worden beschouwd maar een geïntegreerd onderdeel gaan vormen van het gehele besturingssysteem. Enerzijds wordt het voor de gebruiker

daarmee vrijwel onmogelijk deze programma's nog zelf samen te stellen wegens de grote programmeringskosten. Anderzijds zijn in de regel in de gegeven standaardprogramma's reeds zoveel faciliteiten ingebouwd dat er ook weinig behoefte is om het zelf te doen. Een ingebouwde faciliteit kan zijn, ingeval men met geblokte records werkt, dat het 'blokken' en 'ontblokken' reeds in de standaardprogramma's is ingebouwd. Heeft men dan als voorbeeld een index-sequentieel bestand dat men sequentieel wil verwerken dan kan men in het applicatieprogramma volstaan met iedere keer eenvoudig te vragen naar het volgende record terwijl door het standaardprogramma zonodig ook nog gezorgd wordt voor parallel buffering (parallelle verwerking), waardoor automatisch transport van en naar achtergrondgeheugen kan plaatsvinden parallel met het verwerken van gegevens. Aan het gebruik van standaardprogrammatuur is ook nog een ander voordeel verbonden. Wanneer men na verloop van tijd overgaat tot wijziging van de apparatuur, bijvoorbeeld uitbreiding of vervanging van schijvengeheugen, dan behoeft alleen de standaardprogrammatuur te worden aangepast hetgeen in de regel dan een deel van de levering is. Het applicatieprogramma behoeft echter niet te worden gewijzigd.

- Sequentiële verwerking is zonder meer mogelijk bij sequentiële en index-sequentiële bestandsorganisatie waarbij zoals reeds vermeld standaardprogrammatuur alle bijkomstige werkzaamheden als 'blokken', buffering en dergelijke kan verzorgen. In geval van directe bestandsorganisatie hangt de effectiviteit van sequentiële verwerking af van de keuze van de randomizing techniek terwijl in de regel de bijkomstige werkzaamheden als blokken en buffering door eigen programma's moeten worden gedaan.
- Directe verwerking is praktisch onmogelijk met sequentiële bestandsorganisatie, maar wel met index-sequentiële en directe organisatie, daarbij zal men met directe organisatie meestal iets sneller werken dan met index-sequentiële.
- Reorganisatie van het bestand is bij sequentiële organisatie nodig bij iedere mutatie. Bij index-sequentiële en directe organisatie zal het periodiek moeten plaatsvinden, waarbij het bij index-sequentiële meer frequent zal moeten gebeuren dan bij directe organisatie. Daar staat tegenover dat bij index-sequentiële men meestal gebruik kan maken van standaardprogrammatuur terwijl bij directe organisatie men zelf hiervoor zal moeten zorgdragen.
- Gebruikte ruimte in achtergrondgeheugen is bijna optimaal bij sequentiële organisatie. Bij directe organisatie werkt men in de regel met een lagere bezettingsgraad in het primaire gebied dan bij index-sequentiële organisatie. Daartegenover moet men bij index-sequentiële organisatie ook rekening houden met de ruimte voor de index die ongeveer 10% kan bedragen. In beide gevallen komt daar dan nog de ruimte van het overloopgebied bij.

- Ten aanzien van het gebruik van primair geheugen kan het volgende opgemerkt worden:
  Indien men bij index-sequentiële organisatie de indextabellen permanent in het primaire geheugen houdt kan dit een behoorlijke omvang hebben, met het voordeel dat de toegangstijd dan wordt verkleind. Bij directe organisatie is primaire geheugenruimte nodig voor het programma dat de sleuteltransformatie verzorgt.
- Wijziging van de sleutelwaarden betekent bij sequentiële en index-sequentiële een reorganisatie die vergelijkbaar is met normale reorganisaties. Bij de directe bestandsorganisatie kan verandering van sleutelwaarden het gehele oorspronkelijk gekozen systeem onbruikbaar maken. In het algemeen is het vooronderzoek naar voorkomende sleutelwaarden een zeer belangrijke en soms een omvangrijke zaak bij directe bestandsorganisatie. Bij index-sequentiële organisatie is dit veel minder kritisch.
- Variabele recordlengte is geen enkel probleem bij sequentiële organisatie. Bij index-sequentiële kan het enige moeilijkheden geven maar is in de regel toch nog goed uitvoerbaar. Bij directe organisatie geeft variabele recordlengte wel de nodige complicaties doordat hier expliciet van adressen gebruik gemaakt wordt.
- Bestandsbeveiliging tegen calamiteiten is bij sequentiële organisatie betrekkelijk eenvoudig door het grootvader, vader, zoon-systeem. Bij index-sequentiële en directe organisatie en dan in het bijzonder bij directe verwerking van mutaties moet men hieraan de nodige aandacht schenken en meestal programmatisch gezien ook extra maatregelen nemen; dit kan bijvoorbeeld door alle mutaties over een bepaalde periode in een bestand te bewaren, terwijl men na afloop van iedere periode een dump maakt van het gehele bestand. De tijdsduur van de periode is afhankelijk van de toepassing en kan variëren van enkele uren tot enkele dagen.

Met de hier genoemde gezichtspunten voor vergelijking wordt zeker geen aanspraak gedaan op volledigheid. Andere punten komen eventueel vanzelf naar voren indien men in een concreet geval een bepaald bestand met bijbehorende verwerking op verschillende manieren tracht te realiseren.

Index-sequentieel tegenover willekeurig toegankelijk

Teneinde de keuze enigszins te vereenvoudigen volgen hier de belangrijkste algemene voor- en nadelen van indexed-sequential en random bestandsorganisatie. Daarbij is uitgegaan van een vergelijking van indexed-sequential bestandsorganisatie met random bestandsorganisatie waarbij indirecte adressering wordt gebruikt. Indien de random organisatie met behulp van directe adressering eenvoudig en efficiënt kan worden toegepast, is dit te prefereren.

De voordelen van de indexed-sequential bestandsorganisatie zijn: een compacter gebruik van de externe geheugencapaciteit, eenvoudige sequentiële verwerking (onder meer noodzakelijk voor het vervaardigen van overzichten) en een eenvoudige toepassing bij het in gebruik nemen van een computersysteem, omdat standaardsoftware veelal aanwezig is. Een uitgebreide analyse van de eigenschappen van de identificaties voor het kiezen van een adresberekeningsmethode is niet nodig.

De nadelen van de indexed-sequential bestandsorganisatie zijn: een minder snelle toegankelijkheid tot een record bij het verwerken van random (niet gesorteerde) transacties. Tevens zijn omvangrijkere procedures nodig voor het onderhouden van bestanden. De indextabellen dienen daarbij ook aangepast te worden.

Voordelen van de random bestandsorganisatie zijn: het sneller verwerken van de transacties en de eenvoudige procedures voor het onderhouden van bestanden (geen reorganisaties normaliter, geen indextabellen).

Nadelen zijn: een minder efficiënt gebruik van het schijvengeheugen, voor sequentiële verwerking (overzichten) moet een ketting bijgehouden worden ('gehele file on line') of een sorteerslag uitgevoerd worden.
Voorts maken dynamische bestanden de eenmaal gekozen adresberekeningsmethode snel minder efficiënt.

We kunnen concluderen dat de snelheid waarmee een transactie 'at random' verwerkt wordt, een van de belangrijkste verschilpunten is tussen een indexed-sequential en een random bestandsorganisatie. Vaak is het gewenst de voordelen verbonden aan het werken met de indexed-sequential organisatiemethode te combineren, zodat daarbij toch een snellere verwerking van randomtransacties wordt verkregen. Er zijn verschillende mogelijkheden om dit doel te bereiken, waarbij vooral de beschikbare hoeveelheid werkgeheugen van de computer een belangrijke rol speelt. Wanneer er voldoende geheugencapaciteit beschikbaar is, kan de cilindertabel tijdens gehele computerbewerkingen worden opgenomen en geraadpleegd in het werkgeheugen. Het lees/schrijfmechanisme van het externe geheugen is steeds en uitsluitend beschikbaar voor de records in het te verwerken bestand. Dit mechanisme behoeft niet steeds heen en weer te bewegen tussen het bestand en de cilindertabel in het schijvengeheugen. Dit levert een aanzienlijke tijdwinst op.

Een andere methode om een snellere procedure bij een bepaalde toepassing te krijgen is het gebruiken van een combinatie van de random en de indexed-sequential bestandsorganisatie. In sommige

bestanden heeft ongeveer 80% van de mutaties betrekking op slechts 20% van de bestandsrecords. Dit bestand kan geheel of gedeeltelijk indexed-sequential georganiseerd worden, waarbij echter tevens voor 20% van de records (de zeer frequente muterende), een afzonderlijke adrestabel wordt opgebouwd. Deze tabel bevat voor elke identificatie zowel de gegevens voor deze identificatie als het bijbehorende schijvenadres. Hij wordt tijdens de verwerking van het bestand geheel in het werkgeheugen van het computersysteem opgenomen en aldaar geraadpleegd. Hierdoor kan nu voor 80% van de transacties het bijbehorende schijvenadres direct worden bepaald. Een nadeel van deze methode is dat het onderhouden van het bestand door het gebruiken van deze extra tabel omvangrijker wordt. Dit nadeel is bij zeer statische bestanden echter beperkt. Het voordeel van deze methode is een snellere verwerkingsprocedure dan met de normale indexed-sequential methode.

# 6 BESTANDSONTWERP

Als aansluiting op het vorige hoofdstuk en als een van de noodzakelijke gegevens bij het bestandsontwerp presenteren we hier nog eens een beknopte vergelijking tussen de sequentiële, index-sequentiële en direct toegankelijke bestandsorganisatie.
Figuur 23 vergelijkt ze ten aanzien van de basis-bestandsoperaties. In het algemeen kan men stellen, dat het zoeken van een record (en daarna er iets mee doen) het snelst opgelost wordt door de directe bestandsorganisatie. Men kan zich echter ook voorstellen, dat veel records zoeken aan de hand van hun sleutels toch ongeveer neerkomt op het hele bestand doorwerken, zodat voor batchgewijs verwerken de sequentiële organisatie zeer goed zo niet de beste is (zie figuur 24).

Opgave: Ga na of de tabel in figuur 22 correct is.

| | seq. org. | index-seq. org. | directe org. |
|---|---|---|---|
| lage file activity | - | + | + |
| hoge file activity | + | - | - |
| lage file volatility | - | + | ++ |
| hoge file volatility | + | - | - |
| lage file turnover | - | + | + |
| hoge file turnover | + | -- | -- |

-- inefficiënt    ++ zeer efficiënt
\+ efficiënt    - matig

Figuur 22: Vergelijking van de drie bestandsorganisaties ten aanzien van 'bestandsdynamiek'.

Om een optimaal bestandsontwerp te bereiken en om de beste bestandsorganisatie te kiezen, dient men aandacht te schenken aan de punten genoemd in 6.1 tot en met 6.5.

| | Sequentiële organisatie | Index-sequentiële organisatie | Willekeurig (direct) toegankelijke organisatie |
|---|---|---|---|
| Toevoegen van een record | Hele bestand doorwerken (invoegen door copiëren!) - | Deel van het bestand doorwerken: indextabel en soms overflowketen. ++ | Soms een klein deel van het bestand doorwerken: synoniemketting. ++ |
| Verwijderen van een record | Hele bestand doorwerken. - | Deel van het bestand doorwerken: indextabel en soms overflowketen. Record markeren en bij reorganisatie verwijderen. + | Soms een klein deel van het bestand doorwerken: synoniemketting. Veelal markeert men het record en verwijdert bij reorganisatie. + |
| Veranderen van een record | Hele bestand doorwerken. - | Record zoeken via indextabel, soms overflowketting aflopen. + | Record direct vinden: soms synoniemketting aflopen. ++ |
| Veranderen van sleutelwaarden | Hele bestand doorwerken. - | Hele bestand doorwerken (reorganisatie). - | Hele bestand doorwerken: (reorganisatie) nieuw adreseringsalgoritme ontwerpen. -- |
| Zoeken van een record | Hele bestand doorwerken. - | Deel van het bestand doorwerken: indextabel en soms overflowketen. + | Direct; soms een klein deel van het bestand doorwerken: synoniemketting. ++ |

- : inefficiënt, + : efficiënt, ++ : zeer efficiënt.

Opm.: Wanneer men bij 'index-sequentieel' of 'direct toegankelijk' niet meteen het record vindt, zal men verder moeten zoeken (overflow, synoniemen). In deze tabel is een kettingadressering hiervoor verondersteld. Dit hoeft natuurlijk niet.

Figuur 23: Vergelijking van de drie bestandsorganisatievormen ten aanzien van de 'basisbewerkingen'.

| | Sequentiële organisatie | Index-sequentiële organisatie | Willekeurig toegankelijke organisatie |
|---|---|---|---|
| Sequentiële verwerking | Efficiënt; hele file hoeft niet "on line" te zijn. ++ | Gaat; hele file hoeft niet "on line" te zijn. + | Is mogelijk; efficiency hangt af van adresseringsalgoritme; file moet "on line" zijn. - |
| Postgewijze verwerking; (directe verwerking) | Zeer inefficiënt; wordt vrijwel nooit toegepast. -- | Redelijk efficiënt, als indextabellen niet te lang zijn. Als volume-index en track-index in het kerngeheugen passen, werkt het snel. + | Gaat goed. ++ |
| Frequentie van reorganisatie | Geschiedt bij elke mutatie (men copieert al muterend naar een nieuw bestand). - | Periodiek noodzakelijk, als overflowgebied al te vol raakt. + | - Een enkele maal nodig. (Synoniemketens herordenen; veel gevraagde records voorin.) - Als sleutelwaarden veranderen, dan is bovendien nieuw adresseringsalgoritme nodig. ++ |
| Variabele recordlengte | Gaat zeer goed. ++ | Gaat. + | Compliceert adresseringsalgoritme; komt niet veel voor. - |
| Geheugengebruik | Efficiënt. ++ | Tabellen kosten ± 10%. Primaire gebied niet te vol ± 90%. Overflowgebied is ook nodig. + | Adresseringsalgoritme kost ruimte. Primaire gebied slechts tot 80% bezetten. Ruimte voor synoniemen is ook nodig. -- |
| Gangbare standaardsoftware voor bestandsorganisatie (b.v. voor "blocking", "parallel buffering"). | Bij vrijwel alle huidige systemen aanwezig. + | Bij vrijwel alle huidige systemen aanwezig. + | Lang niet altijd aanwezig. - |

Figuur 24: Vergelijking van de drie bestandsorganisaties ten aanzien van globalere aspecten.

## 6.1 *Gebruikskarakteristieken*

### 6.1.1 Soort verwerking

De keuze van de bestandsorganisatie is zeer sterk afhankelijk van de soort verwerking. Deze verwerking kan sequentieel of random zijn. Opvragingen van gegevens kunnen gebeuren via een sleutel of op sleutel tezamen met andere gerelateerde data of op iteminhoud. Het ontwerpen van bestanden zal dan ook naar de aard van het soort systeem waar ze gebruikt worden, andere problemen geven.

Voorbeelden:

Produktiesector
De relaties binnen en tussen bestanden, die in de produktiesector gebruikt worden, zijn meestal tamelijk gecompliceerd. De creatie van een produktstructuurbestand, de materiaalbehoefte berekeningen, het bijhouden van een stuklijst, zijn enkele van de toepassingen in deze sector.
Er bestaan echter uitgebreide standaard softwarepakketten voor het opzetten, bijhouden en raadplegen van bestanden in deze sector (zoals BOMP van IBM, PRINSYS van Philips etc.).

Informatieve servicesector
Het raadplegen van informatie uit een (openbare) 'service data bank' is zeer gecompliceerd, doordat ieder item of combinatie van items binnen een record, sleutel kan zijn waarop de informatie geraadpleegd wordt. Dit probleem vindt men in allerlei bibliotheken of registers zoals: personeelsafdelingen, medische gegevens, bevolkingsregisters, etc.
Ook hier komen de laatste tijd steeds meer standaardpakketten waarmee dergelijke bestanden opgezet en geraadpleegd kunnen worden.

Distributiesector
Deze sector vereist meestal slechts eenvoudige relaties tussen de bestanden, maar door de vele kleine variaties in de diverse toepassingen in deze sector, is het moeilijk om hiervoor standaardpakketten te ontwerpen. Veel eigen ontwerp en programmering zal dan ook in deze sector toegepast worden, natuurlijk zoveel mogelijk gebruik makend van standaard software (voor 'blocking' bijvoorbeeld).

### 6.1.2 Grootte van het bestand

Voor omvangrijke bestanden kan het noodzakelijk zijn om dergelijke bestanden te splitsen. Deze techniek wordt vaak aangeduid met de term splitting files.
Redenen voor het splitsen van het bestand kunnen zijn:

- de noodzaak records te kunnen benaderen via andere itemwaarden dan de sleutel van het record (inverted file);
- indien bepaalde delen van een record slechts zelden wijzigen of opgevraagd worden kan men overwegen de 'zelden gerefereerde recorddelen' in een apart bestand op te nemen (processing cycle files);
- indien de configuratie (bijvoorbeeld het kerngeheugen) zo beperkt is dat meerdere verwerkingsgangen noodzakelijk zijn kan het splitsen van bestanden I/O-tijd of CPU-tijd verminderen.

Indien records van een bestand bestaan uit delen, die veel gevraagd worden en delen die zelden gevraagd worden, kan men twee behandelingsmethoden volgen:
- Splits het bestand in twee bestanden (bijvoorbeeld een actueel en een historisch bestand). In dat geval spreekt men van processing cycle files of prime en trailer files.
- Splits de records niet, maar haal bij mutatieruns de actieve records uit het hoofdbestand en werk deze bij. Werk vervolgens (met een veel grotere periodiciteit) het hoofdbestand bij met behulp van dit aldus gevormde hulpbestand.
  In dit geval spreekt men ook wel over split files.

### 6.1.3 Verloop van een bestand (file turnover)

Het is van belang om te weten hoeveel records toegevoegd en verwijderd worden in een bepaalde periode.

### 6.1.4 Bedrijvigheid van een bestand (file activity)

Hierbij lette men op:
- het aantal raadplegingen (accesses);
- spreiding van raadplegingen over het gehele bestand;
- records met veel en weinig raadplegingen.

### 6.1.5 Levendigheid van een bestand (file volatility)

Dit is het percentage records, dat betrokken is bij wijzigen (updating).

### 6.1.6 Gebruiksstatistieken

Zelden hebben we de bovengenoemde gegevens ter beschikking, wanneer we een bepaald bestand voor de eerste keer opzetten. Een optimaal bestandsontwerp zullen we daar ook meestal niet bereiken. Om na een aantal jaren een optimaler ontwerp te maken is het noodzakelijk om statistische gegevens vast te houden. Deze statistieken moeten minstens bevatten:
- het aantal toevoegingen en verwijderingen uit een bestand per periode;

- het hoogste aantal verbonden (keten)records behorende bij een masterrecord over een bepaalde periode;
- het aantal raadplegingen van elk record afzonderlijk over een bepaalde periode.

Het bijhouden van deze statistieken heeft natuurlijk extra verwerking en plaats nodig, maar wanneer daardoor na een kortere of langere tijd een optimaal bestandsontwerp bereikt kan worden, is het zeer zeker de moeite waard.

## 6.2 *Soorten bestanden*

Hoofdbestand (master file of permanent bestand)
Dit is een bestand, waarin wel wijzigingen optreden, maar dat toch grotendeels intact blijft gedurende een tijd, die lang is vergeleken met de tijd tussen twee bewerkingen van het bestand.

Gerelateerd bestand (related file)
Een gerelateerd bestand bevat records die gekoppeld zijn aan records uit een hoofdbestand.

Geïnverteerd bestand (inverted file)
Een geïnverteerd bestand bestaat uit records waarin een bepaalde sleutel is opgeborgen tezamen met verwijsadressen naar records in één of meerdere hoofdbestanden die alle aan een bepaald criterium voldoen.

Transactiebestand (transaction file) of inputbestand (mutatiebestand)
Het transactiebestand bevat de input voor het systeem. Deze input zal gebruikt worden om hiermee hoofdbestanden te raadplegen of bij te werken (inquiry en activity records).

Outputbestand
Zoals de naam reeds zegt worden dergelijke bestanden gecreëerd om als inputbestanden te dienen voor andere systemen of verdere verwerking binnen een systeem. Een bijzondere vorm van een outputbestand is een 'print'bestand dat als verdere verwerking alleen maar afgedrukt wordt in een bepaald formaat.

Historisch bestand (history file)
Als historische bestanden kunnen beschouwd worden vroegere versies van hoofdbestanden of transactiebestanden, nodig om het actuele hoofdbestand te kunnen reconstrueren in geval van een calamiteit. Men spreekt ook van historische bestanden wanneer men gegevens bewaart die op latere periodieke tijdstippen verwerkt worden.

Een bijzondere vorm van een historisch bestand is het zogenaamde totaalbestand (summary file). Op dergelijke bestanden worden overeenkomstige gegevens uit meerdere periodes gecumuleerd om als historische gegevens bewaard te blijven.

'Recovery' of 'Back-up' bestand

(Back-up bestand is een vroegere image van het bestand.)
Zoals de naam reeds zegt, zal een dergelijk bestand alle noodzakelijke gegevens bevatten om de verwerking te herstarten wanneer er een hardware of software calamiteit opgetreden is.
Tijdens de verwerking wordt soms ook een log-file aangemaakt. Dit bestand bevat in feite niets anders dan de reeds verwerkte mutaties. Met behulp van de 'back-up' en de log-file kan het bestand altijd weer in de toestand worden hersteld die het bezat op het moment van bovengenoemde calamiteit.

Data Base

Een data base is een centrale opslag van gerelateerde gegevens die binnen een informatiesysteem gebruikt worden en die ter beschikking staan voor meer dan één gebruiker.
De gebruiker van een data base systeem heeft in zijn programma's in wezen niets te maken met de fysieke organisatie van een data base. De gegevens die hij nodig heeft worden hem door de 'Data Base Handler' ter beschikking gesteld. Via een 'Data Manipulation Language' kan hij zijn verzoeken om gegevens aan deze DBH kenbaar maken.

Primair (prime) en trailer bestand

Wanneer de items in een record verdeeld kunnen worden in veel en weinig gebruikte, kan het bestand in delen gesplitst worden. Het bestand met de veel gebruikte items wordt het primaire bestand genoemd. De resterende items van het record worden dan in een trailer bestand opgeborgen. Beide bestanden hebben alle sleutels van het oorspronkelijke bestand.

Partitioned bestand

Een partitioned bestand bestaat uit onafhankelijke groepen van sequentieel georganiseerde records, members genoemd. Via een catalogus (directory) worden de namen van members van deze file vastgelegd.
Dergelijke bestanden worden vaak gebruikt bij programma- of tekstbibliotheken.

## 6.3 *Gegevensbepaling*

De eerste stap in het ontwerpen van een bestand is het vaststellen

welke informatie het moet gaan bevatten. In de tweede stap worden de procedures, die van het bestand gebruik zullen gaan maken, vastgesteld.
Leg van elk van deze procedures vast van welke gegevens ze gebruik maken, hoe vaak en welke volgorde etc.
Hierbij moeten we letten op de volgende punten:
- Ga voor elk inputgegeven na of de gegevens waaraan het refereert in andere bestanden, inderdaad aanwezig zijn.
- Weeg het effect van het opslaan van een bepaald gegeven tegenover het bijvoorbeeld elke keer opnieuw berekenen.
- Reserveer ook plaats voor gegevens die tijdens verwerking ontstaan, indien we die gegevens in een later stadium weer nodig hebben (bijvoorbeeld eindsaldo van een debiteur te gebruiken als beginsaldo voor volgende periode).
- Onderzoek alle applicaties die van dit bestand gebruik maken, om de mogelijkheid van het vergeten van noodzakelijke gegevens te verhinderen.
- Ga de huidige applicaties op hun toekomstige eisen na; dit zou kunnen betekenen dat we nu al plaats reserveren voor de dan benodigde items.
- Doe hetzelfde ook voor toekomstig geplande applicaties. Het zal veel gemakkelijker zijn om nu één of enkele extra velden te reserveren, dan om later de hele bestanden te reorganiseren.
- Bestudeer de mogelijkheid om bestaande bestanden te consolideren in één bestand, om zodoende duplicering van gemeenzame gegevens te elimineren (file consolidation). Hierbij moeten we natuurlijk er wel goed op letten dat de verwerkingstijden niet te veel omhoog gaan.
- Overtuig u ervan dat de gegevens die in een nieuwe applicatie nodig zijn, waarvoor het betreffende bestand ontworpen wordt, niet reeds aanwezig is in een bestaand bestand.
- Test het bestand, wanneer het opgezet is, of het voldoet aan de eisen die gesteld worden door de applicaties, die dit bestand gebruiken.
- Voeg die items toe die om technische redenen nodig zijn, zoals extra items in variabele lengterecords.
- Denk ook aan bestandsnazorg (file maintenance) en interne controle eisen.

## 6.4 *Recordontwerp*

### 6.4.1 Veldontwerp (field design); fysieke eigenschappen

Een veld moet altijd zo groot zijn, dat het grootst mogelijke aantal tekens dat opgeborgen moet kunnen worden in dat veld er ook inderdaad in past. In sommige gevallen kan men door speciale technieken

hiervan afwijken (record compaction). Wanneer men bijvoorbeeld weet dat de eerste twee posities van een ordernummer altijd nul zijn, kan men bij het opbergen van een ordernummer in het betreffende veld deze nullen ook weglaten. Omdat bepaalde media zoals bijvoorbeeld ponskaarten en schijven (sector, spoor) een vast aantal posities bevatten is het zeer essentieel om deze limiet in het oog te houden om efficiënte en praktisch bruikbare records te ontwerpen.
Bij de lengte van velden is ook van belang de soort apparatuur waarmee deze gegevens verwerkt worden.
Dit geldt vooral wanneer we werken met vaste woordlengte computers waarbij niet elke positie van het geheugen adresseerbaar is, maar slechts groepen van posities. Dit heeft consequenties ten aanzien van het opbergen van alfabetische en alfanumerieke gegevens.

Bij variabele woordlengte machines of machines die zowel met vaste en variabele woordlengte kunnen werken heeft men nagenoeg geen restricties ten aanzien van de veldlengte en inhoud van de velden.
Altijd geldt dat de velden niet groter moeten zijn als nodig is, dit beïnvloedt de recordlengte en daardoor ook de benodigde geheugencapaciteit en de benodigde accesstijden. Hierbij moet men echter wel zeer goed op toekomstige eisen letten, want het is altijd veel eenvoudiger om een veld, kijkend naar de toekomst, nu bijvoorbeeld al een positie groter te maken, dan om naderhand het hele bestand te reorganiseren, hetgeen ook herprogrammering van de bestaande applicaties inhoudt.
Omdat een verkleining van een record altijd positieve resultaten te zien zal geven ten aanzien van geheugencapaciteit en access, moeten we altijd inventariseren of niet bepaalde 'field compaction' technieken toegepast kunnen worden zoals:

- Decimal scaling

Hierbij worden van decimale getallen alleen de significante posities opgeborgen, het vaste aantal nullen voor of achter deze significante posities wordt aangegeven door een positieve respectievelijk negatieve scalingfactor.
Voorbeeld:

| Variabele | Niet 'gescaled' getal | 'Scaled' getal | Scaling factor |
|---|---|---|---|
| X | .00123 | .123 | -2 |
| Y | 100.00 | .100 | 3 |
| Z | .987 | .987 | 0 |

- Drijvende kommagetallen (floating point)

Bij decimal scaling was het aantal scaling posities altijd vast terwijl bij drijvende komma het aantal te onderdrukken nullen voor elk getal volkomen variabel is.

De significante posities worden in de mantissa opgegeven terwijl de exponent het aantal voorafgaande of volgende nullen bevat.

| Exponent | Teken | Mantissa |
|---|---|---|

- Weglating van de hoogste positie

Wanneer de maximum getalwaarde voor een bepaald gegeven bijna nooit bereikt wordt zou men kunnen overwegen om de hoogste positie weg te laten. Zou het in een bepaald geval toch gebeuren dat het op te bergen getal groter is dan de veldlengte, dan krijgen we automatisch een 'overflow' conditie van de computer, die afgetest kan worden en een overflow subroutine, die uitgevoerd kan worden.

- Variabele lengtevelden

Alleen de significante tekens worden opgeborgen, de scheiding tussen de diverse velden gebeurt door middel van speciale tekens. Wanneer we werken met variabele lengtevelden krijgen we meestal ook variabele lengterecords, hetgeen van belang is bij de te gebruiken bestandsorganisatie en daarmee samenhangende standaardinvoer/uitvoer programmatuur (Data Management).

- Bitting

Hierbij heeft ieder 'bit' in een of meer octaden een eigen betekenis of controlefunctie.
Voorbeeld: Binnen één octade kan bit 1 representeren of actief of inactief, bit 2 en 3 uit één of vier mogelijke kredietklassen, bit 4 en 5 uit één of vier leeftijdclassificaties.

- Coding

Coderen kan gebruikt worden om grotere velden te vervangen.
Voorbeeld: Een één positie code kan aangeven de maateenheid zoals 1 voor dozijn, 2 voor ons, 3 voor kilo, etc.

- Heading

Hierbij worden alle gemeenschappelijke data voor meerdere records slechts eenmaal opgenomen, gevolgd door de respectievelijke detail-items.

- Substituering

Hierbij worden bepaalde tekens of groepen van tekens vervangen door andere, natuurlijk met als doel het aantal posities kleiner te maken.
Voorbeeld: Normaliter zal de maand voorgesteld worden in een tweecijferige code van 01-12. Wanneer we echter de 10 vervangen door 0 en de 11 en 12 door bijvoorbeeld A en B komen we met één-'cijferige' code toe.

- Vormen van field compaction zijn ook nog het gebruik van binaire, packed-decimal velden en hexadecimale notatie.

Wanneer we een bepaalde 'compaction' techniek willen gebruiken moeten we de volgende factoren niet uit het oog verliezen:

a. Hoeveelheid werkgeheugen dat de extra routines kosten om de codeer- en decodeerfuncties uit te voeren.
b. De ervoor benodigde tijd.
c. Het compaction percentage dat bereikt wordt.
d. Compatibiliteit met programmeersystemen.
e. Behouden van de 'collating' volgorde.
f. Behouden van vaste veldlengte.
g. Het effect op het totale systeem inclusief de handmatige werkzaamheden.

### 6.4.2 Bepaling van de volgorde van de gegevens

De volgorde van de gegevens is het meest kritisch voor die bestanden die gecreëerd worden vanaf documenten, zoals ponsen van kaarten en papertape, terminal operaties, en dergelijke. Alles wat deze werkzaamheden kan vereenvoudigen verzekert een snellere en juistere creatie.
Op de volgende punten moet hierbij gelet worden:

- Zoveel mogelijk vastleggen van de gegevens in dezelfde volgorde als ze voorkomen op het document.
- Velden met overeenkomstige inhouden in meerdere bestanden zoveel mogelijk op dezelfde plaats in de records vastleggen, dit vergemakkelijkt de sortering en controle van dergelijke bestanden.
- Sorteervelden aansluitend achter elkaar zetten ('minor' sorteerveld rechts en opklimmend tot 'major' links aangesloten).
- Compatibiliteit met de data karakteristieken van de te gebruiken computer.
- Het bij elkaar zetten van alfabetische/alfanumerieke gegevens, vooral in vaste woordlengte machines.
- Bij gebruik van variabele lengterecords de velden die altijd voorkomen altijd op dezelfde posities houden.

### 6.4.3 Bepaling van het recordformaat en blokking

Bij het selecteren van het recordformaat en de blokking spelen de volgende factoren een belangrijke rol:

- Limiet van het bestand

Papieren ponsband en magnetische band zijn media waarop de gegevens in bijna ongelimiteerd grote records kunnen worden vastgelegd (hierbij natuurlijk niet lettend op restricties van het werkgeheugen etc.).
Ponskaarten hebben een limiet van 80 tot 90 kolommen terwijl magnetische schijven, trommels en dergelijke ook restricties kennen door hun opbouw in sectoren, sporen en cilinders. In theorie is het natuurlijk mogelijk om records op dergelijke media te schrijven die groter zijn dan één spoor of cilinder maar we veroorzaken hierdoor

zoveel extra programmaroutines en verwerkingstijd, dat het op praktische gronden af te raden is.
Wanneer we dan ook een record ontwerpen dat op dergelijke media vastgelegd wordt, moeten we op de volgende punten letten:

- Kies een zodanige recordlengte, dat wanneer deze records geblokt worden, het blok nog altijd goed binnen een spoor past.
- Beperk de maximale lengte voor variabele lengterecords en blokken ook tot een spoor.
- Wanneer we werken met magnetische schijven waarvan de sporen ingedeeld zijn in sectoren, kunnen we meerdere sectoren met één opdracht lezen of schrijven zolang we binnen één cilinder blijven. Bij het ontwerp van het bestand moeten we er voor zorgen dat we niet een aantal sectoren lezen die op meerdere cilinders staan. Dit kunnen we controleren door het aantal sectoren dat we in één opdracht lezen of schrijven te delen op het berekende aantal cilinders van het bestand. Het resultaat moet een geheel even getal zijn.

- Eisen te stellen aan het werkgeheugen

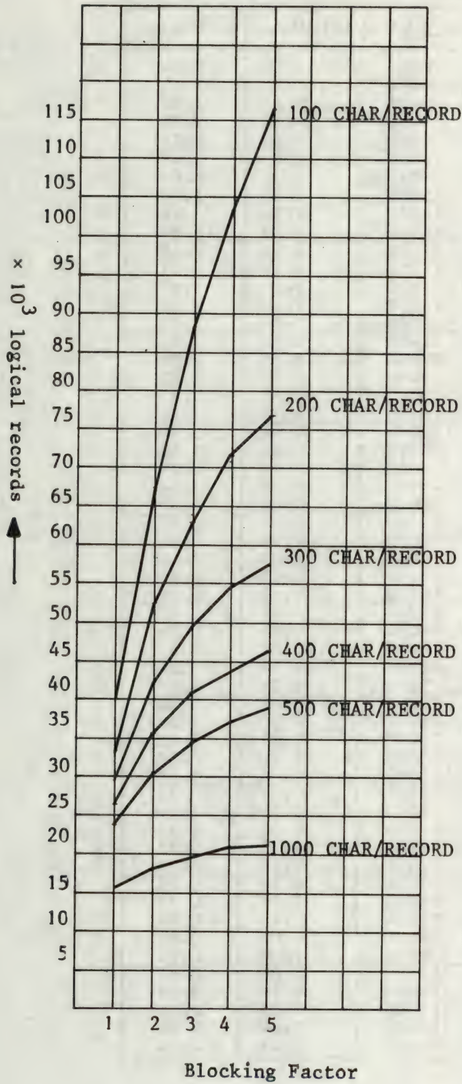
Aangezien door 'Data Management Software' alleen fysische records gelezen en geschreven worden, moet er in het werkgeheugen een gebied gereserveerd zijn groot genoeg om een dergelijk fysisch record (blok) te bevatten. Voor een efficiënte verwerking is het meestal raadzaam om meerdere van dergelijke I/O gebieden te reserveren. Buiten deze I/O gebieden is er meestal ook nog een werkgebied nodig ter grootte van een logisch record.

- Capaciteit van geheugenmedia

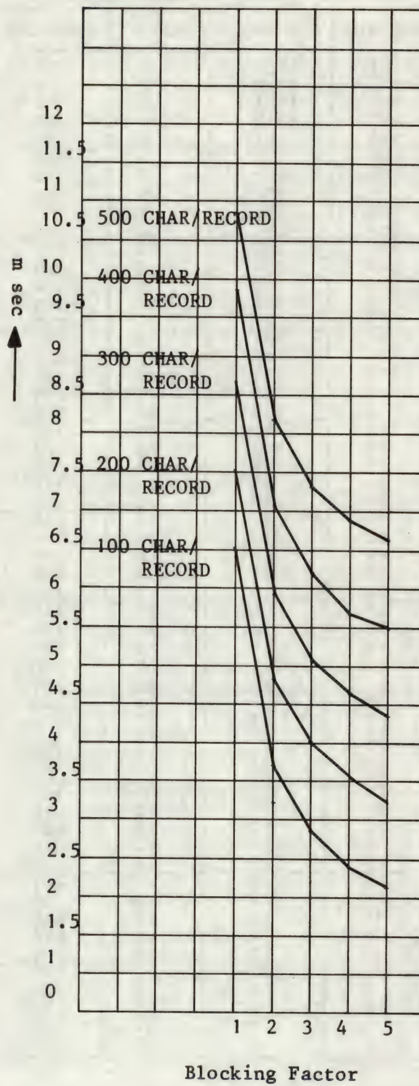
Wanneer we fysische records vastleggen op een magneetband, magnetische schijf en dergelijke zullen er tussen de vastgelegde records niet gebruikte ruimtes ontstaan (interrecord gaps). Dit betekent dat we in geen enkel geval de maximum capaciteit van een geheugenmedium bereiken.
De meeste plaats verliezen we bij het gebruik van ongeblokte records, al moeten we ons tegelijkertijd realiseren dat ongeblokte records het gemakkelijkst te verwerken zijn. Door records te blokken verhogen we de bruikbare capaciteit maar we verhogen de lees/schrijftijd per fysisch record. We winnen echter weer tijd per logical record binnen zo'n blok.
In figuur 25 en 26 kunt u een voorbeeld zien van de invloed van blokken op de capaciteit en de gemiddelde accesstijd. In figuur 27 in een andere vorm hetzelfde voor magnetische schijf.

- Systeemprogrammatuur ondersteuning

Elke standaard 'Data Management' ondersteunt bepaalde recordformaten en wanneer we niet van plan zijn om eigen programmatuur te ontwikkelen, zullen we ons bij de bepaling van het recordformaat moeten houden aan de toegestane. Hetzelfde geldt ook ten aanzien van andere standaardprogrammatuur die we willen gebruiken zoals bijvoorbeeld sorteerprogramma's.




Figuur 25: Relatie tussen blokking en de capaciteit van het bestand (tape).




Figuur 26: Relatie tussen blokking en gemiddelde accesstijd (tape).

| BYTES PER RECORD | | RECORDS PER | | | TRANSMISSION TIME IN MS PER RECORD | |
|---|---|---|---|---|---|---|
| MINIMUM | MAXIMUM | TRACK | CYLINDER | MODULE | MINIMUM | MAXIMUM |
| 1741 | 3625 | 1 | 10 | 2000 | 11.16 | 23.24 |
| 1132 | 1740 | 2 | 20 | 4000 | 7.26 | 11.15 |
| 831 | 1131 | 3 | 30 | 6000 | 5.33 | 7.25 |
| 652 | 830 | 4 | 40 | 8000 | 4.18 | 5.32 |
| 533 | 651 | 5 | 50 | 10000 | 3.42 | 4.17 |
| 448 | 532 | 6 | 60 | 12000 | 2.87 | 3.41 |
| 385 | 447 | 7 | 70 | 14000 | 2.47 | 2.87 |
| 335 | 384 | 8 | 80 | 16000 | 2.15 | 2.46 |
| 296 | 334 | 9 | 90 | 18000 | 1.90 | 2.14 |
| 264 | 295 | 10 | 100 | 20000 | 1.69 | 1.89 |
| 237 | 263 | 11 | 110 | 22000 | 1.52 | 1.69 |
| 214 | 236 | 12 | 120 | 24000 | ..37 | 1.51 |
| 194 | 213 | 13 | 130 | 26000 | 1.24 | 1.37 |
| 178 | 193 | 14 | 140 | 28000 | 1.14 | 1.24 |
| 163 | 177 | 15 | 150 | 30000 | 1.04 | 1.13 |
| 150 | 162 | 16 | 160 | 32000 | 0.96 | 1.04 |
| 139 | 149 | 17 | 170 | 34000 | 0.89 | 0.96 |
| 128 | 138 | 18 | 180 | 36000 | 0.82 | 0.88 |
| 119 | 127 | 19 | 190 | 38000 | 0.76 | 0.81 |
| 110 | 118 | 20 | 200 | 40000 | 0.71 | 0.76 |
| 103 | 109 | 21 | 210 | 42000 | 0.66 | 0.70 |
| 96 | 102 | 22 | 220 | 44000 | 0.62 | 0.65 |
| 89 | 95 | 23 | 230 | 46000 | 0.57 | 0.61 |
| 83 | 88 | 24 | 240 | 48000 | 0.53 | 0.56 |
| 78 | 82 | 25 | 250 | 50000 | 0.50 | 0.53 |
| 73 | 77 | 26 | 260 | 52000 | 0.47 | 0.49 |
| 68 | 72 | 27 | 270 | 54000 | 0.44 | 0.46 |
| 64 | 67 | 28 | 280 | 56000 | 0.41 | 0.43 |
| 60 | 63 | 29 | 290 | 58000 | 0.38 | 0.40 |
| 56 | 59 | 30 | 300 | 60000 | 0.36 | 0.38 |
| 53 | 55 | 31 | 310 | 62000 | 0.34 | 0.35 |
| 49 | 52 | 32 | 320 | 64000 | 0.31 | 0.33 |
| 46 | 48 | 33 | 330 | 66000 | 0.29 | 0.31 |
| 43 | 45 | 34 | 340 | 68000 | 0.28 | 0.29 |
| 41 | 42 | 35 | 350 | 70000 | 0.26 | 0.27 |
| 38 | 40 | 36 | 360 | 72000 | 0.24 | 0.26 |
| 36 | 37 | 37 | 370 | 74000 | 0.23 | 0.24 |
| 33 | 35 | 38 | 380 | 76000 | 0.21 | 0.22 |
| 31 | 32 | 39 | 390 | 78000 | 0.20 | 0.21 |
| 28 | 30 | 40 | 400 | 80000 | 0.18 | 0.19 |
| 26 | 27 | 41 | 410 | 82000 | 0.17 | 0.17 |
| 24 | 25 | 42 | 420 | 84000 | 0.15 | 0.16 |
| 22 | 23 | 43 | 430 | 86000 | 0.14 | 0.15 |
| 21 | 21 | 44 | 440 | 88000 | 0.13 | 0.13 |
| 20 | 20 | 45 | 450 | 90000 | 0.13 | 0.13 |
| 18 | 19 | 46 | 460 | 92000 | 0.12 | 0.12 |
| 16 | 17 | 47 | 470 | 94000 | 0.10 | 0.11 |
| 15 | 15 | 48 | 480 | 96000 | 0.10 | 0.10 |
| 13 | 14 | 49 | 490 | 98000 | 0.08 | 0.09 |
| 12 | 12 | 50 | 500 | 100000 | 0.08 | 0.08 |
| 10 | 11 | 51 | 510 | 102000 | 0.06 | 0.07 |
| 9 | 9 | 52 | 520 | 104000 | 0.06 | 0.06 |
| 8 | 8 | 53 | 530 | 106000 | 0.05 | 0.05 |
| 7 | 7 | 54 | 540 | 108000 | 0.04 | 0.04 |
| 5 | 6 | 55 | 550 | 110000 | 0.03 | 0.04 |

Figuur 27: Capaciteit van magneetschijven bij verschillende fysische recordlengtes en gemiddelde transmissietijd van de gegevens.

Extra attentie is vereist wanneer we de bestanden willen gebruiken op verschillende typen computers. Hierbij is niet alleen het formaat van de data records van belang maar evenzeer de formaten van de controlerecords zoals identificatierecords (header-, trailerlabels, etc.).

## 6.5 *Evaluatie van het ontwerp*

Voordat we een bepaald bestandsontwerp definitief maken, moeten we eerst de 'run'tijden en daarmee verbonden kosten voor het gehele systeem berekenen. Het resultaat hiervan moet vergeleken worden met de oorspronkelijke doelstellingen waarvoor het ontwerp is opgezet.

Wanneer tijdens deze evaluatie blijkt dat het systeem input/output gebonden is (dat wil zeggen de I/O-tijd is hoger dan de verwerkingstijd) kunnen de volgende maatregelen overwogen worden:

- Creëer een tweede hoofdbestand dat in de records die velden bevat die niet gebruikt worden in de hoofdverwerking.
  Voorbeeld: Naam, adres, woonplaats uit het klantenbestand worden alleen gebruikt tijdens het afdrukken van de fakturen. Hiervan zou een apart NAW-bestand opgezet kunnen worden.
- Wanneer we met magneetbanden werken, zouden we voordat we met de feitelijke verwerking beginnen, de te verwerken records in een aparte job van het bestand kunnen afsplitsen, eventueel tegelijkertijd met het terugschrijven van de bijgewerkte records van de vorige run.
- Wanneer steeds slechts een beperkt aantal records actief is, splits het bestand in twee bestanden, één met de veel geraadpleegde records en het andere met de overige records.
- Verhoog het aantal inputbuffers. Hierdoor bereiken we een grotere overlap van I/O-tijden en verwerkingstijd.

# 7 CONTROLE VAN HET BESTAND (FILE CONTROL)

Het ontwerp van een bestand kan niet los gezien worden van de omgeving waarin het betreffende bestand moet functioneren.
In de volgende paragrafen zullen we enige controle en 'maintenance' maatregelen bespreken, die niet alle rechtstreeks betrekking hebben op bestanden maar zonder welke de bestanden niet goed kunnen functioneren en beschermd worden.

## 7.1 *Interne controle*

Algemeen

Het gaat om de administratieve interne controle. Het betreft dus dat deel van het totale gebied der interne controle dat gericht is op de verzekering van de juistheid van de administratieve registratie en de berichtgeving omtrent het bedrijfsgebeuren.

Wil de verantwoordelijke leiding voor de juistheid instaan van de verantwoording en zekerheid geven dat een juiste basis aanwezig is voor de beslissingsvoorbereiding dan moet de gegevensverwerking aan een aantal voorwaarden voldoen. Zekerheid moet verkregen worden over:
- juist functioneren van systeem;
- volledig verwerken van alle gegevens;
- juiste basis voor beslissing;
- inhoud van bestanden.

Mate van controle

De controlemaatregelen worden beïnvloed door:
- systematiek van de gegevensverwerking;
- toleranties die de leiding wil accepteren;
- doelmatigheidsoverwegingen.

In een concreet geval zal de behoefte aan beveiliging moeten worden gedefinieerd. Hiervoor zal een keuze gemaakt moeten worden uit beschikbare controlemethodieken.

### Plaats van controle

Bij de informatieverwerking kan men een aantal fasen onderkennen, welke ook in het controleproces een belangrijke rol spelen, bijvoorbeeld:

- de invoer;
- de verwerking door de computer;
- het gebruik van de geproduceerde gegevens;
- de correctieprocedure;
- maintenance systeem.

Een belangrijk gedeelte van de controle zal plaats moeten vinden in de computerprocedure, waardoor automatisch een berichtgeving ontstaat omtrent de door de computer geconstateerde afwijkingen van vooraf vastgestelde normen. Indien de gesignaleerde afwijkingen een gevolg zijn van fouten in de standaard software of verkeerde bediening van de computer zijn deze van speciaal belang voor het verwerkingscentrum. Indien zij echter betrekking hebben op fouten in de ingevoerde gegevens of de programmering zelf, zijn deze van belang voor degene die verantwoordelijk is voor het systeem van informatieverwerking. De geconstateerde fouten zullen daarom, afhankelijk van de soort fout, moeten worden nagegaan door speciaal daarvoor aangewezen functionarissen van het verwerkingscentrum, of van de voor het systeem verantwoordelijke afdeling.
Van groot belang is ook dat de als onderdeel van een systeem benodigde correctieprocedures op zichzelf zeer goed gecontroleerd worden.

### Testen van de controle

Het is zeer belangrijk dat tijdens het testen van het programma ook de juiste werking van de ingebouwde en geprogrammeerde controles wordt nagegaan, iets wat in de praktijk nog al eens (gedeeltelijk) wordt vergeten. De testgevallen zullen dus bewust fouten moeten bevatten om ook de controles te onderzoeken.
Het is ook noodzakelijk om een gedeelte van de testgevallen en de testproduktie door iemand anders dan de programmeur te laten samenstellen.

### Accountantsdienst

Het verdient aanbeveling om met de accountant onder wiens controle het informatiesysteem valt, van te voren overleg te plegen omtrent de wensen welke hij eventueel kan hebben over in het systeem aan te brengen controles.

Reconstructie na fouten

Tot de controlemaatregelen kunnen ook worden gerekend die, welke nodig zijn om in geval van onverhoopt verlies de verloren gegane gegevens te reconstrueren. Hiertoe wordt meestal de grootvader-vader-zoon-conceptie gebruikt. De zoon bevat de nieuwe stand, de vader de vorige stand en de grootvader de weer vóór de vader liggende stand. Ingeval nu bij verwerking van mutaties bijvoorbeeld de oude (= vorige bestand = vader) wordt beschadigd, kan door middel van de bewaarde grootvader en de mutaties van de vorige verwerkingsrun de vader worden gereconstrueerd.
Bij het gebruik van magneetband ligt de hier genoemde conceptie voor de hand, omdat daar nooit het oude bestand wordt overschreven, maar een nieuw bestand op een nieuwe band wordt opgebouwd. Bij het gebruik van andere hulpgeheugens worden van het oude bestand alleen de gemuteerde standen gewijzigd. Hier wordt dus de vader tot zoon gemaakt en kan men de oude stand bewaren door bijvoorbeeld vóór de verwerking de oude stand op een magneetband te dumpen. Indien het aantal gemuteerde standen niet al te groot is, kan een snelle reconstructie worden verkregen door tijdens de verwerking de gemuteerde standen ook op een magneetband te plaatsen. Gaat er iets mis, dan wordt eerst de inhoud van de dumptape in het hulpgeheugen geplaatst en daarna de gemuteerde standen; er is dan dus geen herhaalde bewerking van de gegevens nodig.

## 7.2 *Ingebouwde controles*

Onder ingebouwde controles worden niet alleen verstaan de door de computer verrichte automatische controles op de juiste werking van de hardware, maar ook de controles welke in door de fabrikant geleverde en eventueel in eigen bedrijf ontwikkelde standaardprogramma's zijn vastgelegd.
Deze controles worden ook wel 'machine gerichte controles' genoemd.

### 7.2.1 Hardware

Hardware controles kunnen worden onderverdeeld in de volgende vier soorten:

- Controle op de juiste werking van de in- en uitvoereenheden. Hiertoe behoren bijvoorbeeld: dubbel lezen (ponskaart- en ponsbandlezer), hole count (ponskaartlezer en -ponser), echo check, papier op (printer), lezen na schrijven (magneetband, hulpgeheugen), gebroken band, begin en einde band, protectiëring (magneetband),

controle of gevonden adres klopt met opgegeven adres (schijvengeheugens en dergelijke).
- Controle op geldigheid en pariteit tijdens in- en uitvoer van gegevens.
- Controle op de juiste werking van de centrale eenheid. Hiertoe behoren bijvoorbeeld: pariteitscontrole bij transport van karakters, geldigheid van de opdrachtcodes, geldigheid van de geheugenadressen, indicatoren voor niet normale condities (overflow, en dergelijke), geheugenprotectie bij multiprogrammering.
- Controle op de goede werking van het gehele computersysteem. Hiertoe behoren bijvoorbeeld: temperatuur en relatieve vochtigheid in de computerruimte, rookontwikkeling in de computerruimte, electriciteitsstoringen, smeltveiligheden, automatische uitschakeling van alle delen van de computer indien genoemde condities gevaar voor beschadiging opleveren.

### 7.2.2 Standaard software

De mate van controle in de standaard software is afhankelijk van de fabrikant en van het gebruik van de software. Door het meegeven van parameters kan de gebruiker veelal zelf bepalen welke ingebouwde controle hij al of niet gebruiken wil. Onderstaand volgt een overzicht van de meest gebruikelijke, in de software als standaard gebouwde controles.

Programma- en bestandsidentificatie
Deze controles behoren tot het zogenaamde 'housekeeping' of initialiseringsgedeelte van het programma. De controle op uitvoering van het juiste programma is soms visueel (uittikken op de consoleschrijfmachine); in andere gevallen, waarbij de programma's door de computer uit een programmabibliotheek (van magneetbanden of schijven) worden gehaald, test het monitorprogramma of wel het juiste programma is ingevoerd.

De controle op de juiste bestanden is iets moeilijker. Er moet niet alleen gecontroleerd worden of het juiste soort bestand aanwezig is, maar ook of het van de juiste datum of periode is (denk aan de grootvader-vader-zoon-conceptie). Het eerste blok met gegevens van een bestand vormt daartoe meestal de 'header-label'. Deze bevat, naast de naam van het bestand, het codenummer van het bestand, de datum van de laatste verwerking of een periodenummer, het sequencenummer (indien het bestand op magneetband meer dan één reel beslaat), de bewaartermijn en het haspelnummer (reelnummer). Als parameters moeten aan het standaardprogramma meegegeven worden het codenummer van het bestand en de datum of periodenummer van de laatste verwerking. Voor de te schrijven magneetbanden bovendien de datum van verwerking. Bij de invoer vanaf ponskaarten of ponsband bevat de eerste kaart of blok een identificatie.

### Volledigheids- en volgordecontrole

Tot de volledigheidscontrole van magneetbanden behoren de zogenaamde block-, record- en hashtotals. Tijdens het schrijven van de magneetband worden tellingen bijgehouden van het aantal geschreven blokken en records en soms tevens van (bijvoorbeeld) de eerste tien posities van elk record.
Deze gegevens worden aan het einde van de verwerking als 'trailer-label' op de magneetband geschreven. Bij het lezen van deze magneetband worden dezelfde tellingen bijgehouden en dan het einde van de verwerking vergeleken met de tellingen in de 'trailer-label'.
Gelijke uitkomsten waarborgen een volledige verwerking van het bestand. Aan het einde van de verwerking blijkt dus of het volledige bestand is verwerkt. Indien dit niet zo is, kan men nagaan welke gegevens niet verwerkt zijn. Om dit te controleren zal men zijn toevlucht moeten nemen tot geprogrammeerde controles.
Bij gebruik van bestanden op schijven en dergelijke is de hierboven beschreven methode alleen mogelijk bij het opbouwen en dumpen van het bestand en niet bij de normale verwerking.

Bij het gebruik van standaardprogramma's voor de in- en uitvoer van gegevens worden ook dergelijke methoden gebruikt. Bij invoer zullen de controletotalen als laatste meegegeven moeten worden door middel van een sluitkaart (ponskaarten) of sluitblok (ponsband). Deze totalen moeten dus tijdens het ontstaan van de gegevens opgebouwd worden. (Opm.: Dit kan ook tot de systeemgerichte controles gerekend worden.)
Ingebouwde volgordecontroles vindt men praktisch alleen bij de standaard sorteerprogramma's. Deze kan op opeenvolging (3 na 3 enzovoorts) of opklimmende of afdalende volgorde geschieden.
De volledigheid wordt na elke sorteerfase gecontroleerd.

### Checkpoint and restart

Hieronder wordt verstaan dat op bepaalde momenten of wanneer bepaalde punten tijdens de verwerking worden bereikt, de gehele staties van de job en de inhoud van het werkgeheugen worden vastgelegd op een hulpgeheugen, meestal een magneetband. Indien de verwerking tot dat moment goed was verlopen, dan gaat de computer door tot aan het volgende 'checkpoint'. Is dit niet het geval dan gaat de computer terug naar het vorige checkpoint en begint van daar af weer opnieuw. Een dergelijke procedure deelt in feite een (meestal langdurend) programma in een aantal aparte en onafhankelijke stukken en ieder stuk wordt aan het einde gecontroleerd. Er kan gecontroleerd worden op subtotalen, hashtotalen en dergelijke. Gaat er tussen twee controlepunten iets verkeerd (bijvoorbeeld met de hardware) dan kan ook teruggekeerd worden naar het laatste goede controlepunt. Bovendien biedt deze techniek de mogelijkheid om lang-

durige programma's af te breken voor programma's met een hogere prioriteit, aan het einde van de werkdag, bij niet gepland maar plotseling noodzakelijk onderhoud, etc.

Het herstarten (terugkeren naar het voorgaande controlepunt) kan als volgt geschieden:
- Magneetbanden worden teruggespoeld tot dit voorgaande punt.
- De inhoud van schijven- en andere hulpgeheugens wordt teruggebracht naar de toestand, zoals deze was op het moment van het controlepunt waar naar teruggekeerd wordt. (Om dit te kunnen, moet dus vanaf ieder controlepunt de oude toestand van de gemuteerde standen bewaard worden, bijvoorbeeld op magneetband of een vrij stuk schijfgeheugen.)
- Kaartlezers, printers en dergelijke moeten met de hand aangepast worden.
- Het werkgeheugen moet weer in de toestand, zoals deze was bij het voorgaande controlepunt, hersteld worden.

Schrijf- en leesfouten

Bij het schrijven op een magneetband wordt getest (hardware: read after write) of wel goed geschreven is. Indien dit niet het geval is, wordt de band tot het laatst goed geschreven blok teruggespoeld en wordt opnieuw getracht te schrijven. Dit wordt meestal een paar maal geprobeerd. Lukt het dan nog niet, dan wordt een klein stukje magneetband overgeslagen en daarna pas het blok weggeschreven. Bij het lezen van de band merkt men hier niets van, er is alleen een wat grotere 'inter-record-gap'.
Ook bij het lezen van magneetband worden controles uitgevoerd of deze goed gelezen wordt. In het geval van 'read-errors' wordt ook meermalen getracht het betreffende blok te lezen. Indien dit dan nog niet lukt, kan (door middel van vooraf aan de standaard software meegegeven aanwijzingen) gekozen worden tussen afdrukken van het foute blok op console-typewriter of printer, wegschrijven van het foute blok op een aparte magneetband of domweg overslaan van het blok.
Een andere foutenmogelijkheid is dat het gelezen blok niet de lengte heeft welke vooraf in het programma is opgegeven ('wrong length record'). In dat geval wordt niet herhaald gelezen, wel bestaat dezelfde keuze van behandeling van dit foute blok als bij 'read-errors'.

## 7.3 *Geprogrammeerde controles*

Algemeen

Geprogrammeerde controles zijn één van de belangrijkste vormen van interne controle bij geautomatiseerde informatieverwerking. Enerzijds worden controles geprogrammeerd welke in de plaats

komen van die, welke voorheen op traditionele wijze werden verricht; anderzijds worden geheel nieuwe controles mogelijk welke voorheen om praktische redenen niet werden toegepast.
Geprogrammeerde controles zijn noodzakelijk omdat bepaalde vroegere controles, welke bijvoorbeeld waren gebaseerd op afstemming van tussentotalen, door het meer geïntegreerde karakter van de computerverwerking niet meer mogelijk zijn. Ook de normale menselijke beoordeling van het cijfermateriaal komt te vervallen, waardoor de wijze van verwerking steeds onpersoonlijker wordt.
Bij het maken van een programma mogen de controles niet het sluitstuk vormen, waardoor men bij geheugen- of tijdgebrek controles laat vervallen. Geprogrammeerde controles vormen een wezenlijke noodzaak in het programma.
Geprogrammeerde controles leidden ook (evenals ingebouwde) tot signalen. Deze signalen mogen echter niet dan bij hoge uitzondering leiden tot het stoppen van de computer.
Indien mogelijk, moet de geprogrammeerde controle zodanig worden uitgevoerd, dat, indien niet op een signaal wordt gereageerd, tijdens de volgende run een herhaald signaal wordt gegeven. De controle immers heeft pas zin, indien er op de eventuele signalen ook de nodige actie volgt.

### 7.3.1 Invoercontroles

Programma's, welke speciaal de invoer controleren, worden wel 'screening' programma's genoemd. Er is op dit gebied nauwelijks van enige standaardisatie sprake en dus is er ook geen standaard software voor. Alle hieronder genoemde controles moeten worden geprogrammeerd, behalve soms de reeds genoemde identificatie- en volgordecontrole.

Plaatscontrole
Hieronder verstaat men:
- het controleren van ieder karakter en veld of dat overeenkomstig is met het opgegeven type: numeriek of alfabetisch;
- het controleren van de juiste plaats van ieder veld;
- het controleren van de lengte van ieder veld.

De beide laatste controles zijn vooral van belang indien gewerkt wordt met invoergegevens van variabele lengte.

Limietcontrole
Hiermede wordt bedoeld het testen of een bepaald gegeven niet onder en/of boven een bepaalde norm komt. Indien deze normen niet absoluut zijn, kan het vaststellen hiervan moeilijk zijn. Stelt men ze te nauw, dan volgen er teveel signalen; stelt men ze te ruim dan is er geen voldoende controle. Deze vorm van controle komt ook voor op resultaten van met de computer gedane bewerkingen.

### Volgordecontrole

Naast de reeds in het vorige hoofdstuk genoemde volgordecontroles komen ook volgordecontroles binnen één gegevensgroep voor. Er kunnen bijvoorbeeld meerdere soorten mutaties voorkomen voor een artikel in het voorraadbestand. Wanneer één van die mutaties zou zijn 'blokkeer' dit artikel voor alle mutaties, dan moet deze mutatie ook als eerste aanwezig zijn vóór alle andere mutaties op dit artikel.

### Geldigheidscontrole

Deze kan gebruikt worden om bijvoorbeeld de geldigheid van mutatiecodes te testen. Hierbij moet opgemerkt worden dat altijd op alle mutatiecodes getest moet worden. Zijn er drie mutatiecodes, dan mag niet worden volstaan met slechts op twee te testen en, als deze het niet zijn, maar aannemen dat het de derde is. Ook op deze laatste moet getest worden.

### Afhankelijkheidscontrole

Het komt meermalen voor dat de inhoud van een veld afhankelijk is van één of meer andere velden. Als veld A is ingevoerd dan moet veld B ook gegevens bevatten.

### Volledigheidscontrole

Bij de machine gerichte controles is al genoemd de sluitkaart en dergelijke. De volledigheidscontrole op een gedeelte van de invoer (indien deze zeer veel gegevens bevat) kan worden verkregen door tussentotalen bij de invoer mede te geven. Dit kan een garantie zijn voor volledigheid van alle of een aantal gegevensgroepen, maar behoeft niet aan te duiden dat zij per gegevensgroep volledig is. Indien per gegevensgroep moet worden gecontroleerd kan gebruik worden gemaakt van voorgetelde en bij de invoer meegegeven totalen of hashtotalen per gegevensgroep.
Deze controle weegt vooral zwaar voor tabellen, welke in een programma worden gebruikt en welke vele mutaties ondergaan. Deze tabellen kunnen onderdeel van het programma zijn voor constanten, of door het programma als invoer tijdens de initialisering worden gelezen. In dit laatste geval moet een volledigheidscontrole, bijvoorbeeld in de vorm van een sluitkaart waar het totaal aantal kaarten van de tabel op staat, plaatsvinden. Bij wijziging moet ook deze sluitkaart veranderd worden, zodat ook altijd een controle met de hand plaatsvindt.

### Self-checking codes

Aan kengetallen (sleutelwoorden, keywords, codes) worden soms één of meer controlecijfers toegevoegd, zodat met een redelijke zekerheid de geldigheid van de combinatie kan worden vastgesteld. Indien bijvoorbeeld de ponsmachine voorzien is van een 'self-checking

number device' kan de geldigheid van de geponste code worden vastgesteld. Meestal kan dan slechts één controlecijfer aan de code worden toegevoegd. Indien men een nog grotere mate van zekerheid wil verkrijgen moeten meer controlecijfers worden toegevoegd. De controle kan dan plaatsvinden in de computer.

### 7.3.2 Controle op de verwerking

Enkele van de in de vorige paragraaf genoemde controles kunnen ook toegepast worden tijdens de verwerking. De limietcontrole komt wel eens voor op de uitkomst van een bewerking, vooral bij wetenschappelijke toepassingen. Naast de hieronder genoemde controles hangt het van het systeem af, welke andere eventueel nog geprogrammeerd kunnen worden.

Vierkantscontrole
Een van de oudste bekende controles in de boekhouding is de vierkantscontrole ('crossfooting'). Indien in één lijst een afstemming per regel mogelijk is, moet ook het totaal van alle kolommen afstembaar zijn. Ofwel de som van de totalen per regel moet gelijk zijn aan de som van het totaal per kolom.

Audit trail
Hieronder worden alle documenten en vastleggingen verstaan welke het mogelijk maken vast te stellen hoe een bepaald saldo of totaal tot stand gekomen is of hoe het is samengesteld. Het begrip is ontstaan omdat bij vele computertoepassingen de individuele posten niet meer in het systeem worden bewaard. In de conventionele administratie heeft men veelal de beschikking over een historisch overzicht, waarop alle mutaties zijn vastgelegd.
Het kan noodzakelijk zijn, dat het systeem toch moet voorzien in de mogelijkheden om totalen of saldi op te bouwen uit een vorige stand. Men kan hiertoe aan de hand van de mutaties en de saldi aan het eind van de vorige periode een staat maken (met de computer) waarop staan vermeld oud saldo, mutatie(s), nieuw saldo. Deze methode is nogal duur en zij zou daarom kunnen worden beperkt tot die gedeelten welke belangrijk zijn.
Een andere methode, welke wel leap-frog wordt genoemd, is om bij het afdrukken van een mutatie een verwijzing te geven naar de vorige mutatie. Bij faktureren bijvoorbeeld kan men het faktuurnummer onthouden bij de gegevens van een gemuteerd artikel. Komt er bij een volgende faktuur weer een mutatie van dit artikel voor, dan wordt het onthouden faktuurnummer van de vorige mutatie hierbij afgedrukt en het nieuwe faktuurnummer weer bij de gegevens van dat artikel gevoegd. Op deze wijze ontstaat er een sprongsgewijze verwijzing naar de opvolgende mutaties van het betreffende artikel.

# 8 DATABASES VOLGENS CODASYL CONCEPTIES

## 8.1 *Inleiding*

In moderne bedrijven en organisaties speelt de uitwisseling van informatie - de communicatie - een voorname en algemeen erkende rol. Deze informatie-uitwisseling kan direkt zijn: gesprekken, briefwisseling. Vaak echter komt de informatie-overdracht indirekt tot stand: de informatie wordt vastgelegd en bewaard voor (mogelijk) gebruik op een later tijdstip.

Vooral bij de indirekte informatie-overdracht wordt onder andere automatische informatie-verwerking met behulp van computersystemen toegepast. Hierbij wordt de informatie vastgelegd in de vorm van gegevens (data), welke voor een computer leesbaar en hanteerbaar zijn.

De laatste jaren heeft zich een versnelde ontwikkeling op het gebied van de gegevensverwerking en -opslag voorgedaan; niet zozeer ten aanzien van de technologie van de opslagmedia (afgezien van de prijsontwikkeling), maar ten aanzien van de toepassingen en de daaruit voortvloeiende wijze van denken over gegevensverwerking. Beschouwt men de ontwikkeling van de gegevensverwerking en -opslag dan kan men daarin globaal drie fasen onderscheiden:

1. De eerste fase werd geheel bepaald door de technische mogelijkheden. Zij werd gekenmerkt door sequentiële verwerking en sequentiële opslag omdat de ekonomisch haalbare hardware- en software-technieken slechts deze wijze toelieten (magneetband, ponsband, ponskaart).

2. De tweede fase kenmerkt zich door de introductie van opslag en raadpleegtechnieken, welke typisch gebaseerd zijn op de technologie van willekeurig toegankelijke opslagmedia (disk, drum). De ontwikkeling van bestandsorganisatie-technieken werd sterk bepaald door de toepassingen, maar waren in wezen alle variaties op de index-sequentiële en de directe bestandsorganisatie. Daarnaast bleef uiteraard de sequentiële methode bestaan.

Beide fasen hebben gemeen, dat de toepassingsmogelijkheden zeer sterk bepaald worden door de mogelijke programma- en bestands-

organisatie-technieken. Deze gebruiksmogelijkheden en -beperkingen dringen dan ook sterk door in de organisatie en informatie-verwerking van de uiteindelijke gebruiker.

Ook blijven de programmeurs van een toepassing belast met alle *technische* aspecten van de gekozen bestandsorganisatie. Het gevolg is geweest dat bestanden in het algemeen voor individuele toepassingen ontworpen zijn waarbij de structuur van het bestand en de gekozen wijze van verwerking doorgedrongen zijn in het applicatieprogramma. Indien gegevens voor andere toepassingen nodig zijn, kan men kiezen uit twee mogelijkheden:

a) Copiëren van de gegevens uit het bestand in een eigen bestand, waarvan men de structuur en verwerkingswijze zelf bepalen kan.

b) Het bestaande bestand gebruiken, waarbij de vrijheid van programmeren danig beperkt wordt door de vastgelegde opzet van het bestand.

Hoewel op dit moment vele gerealiseerde en hier en daar ook te realiseren geautomatiseerde systemen voor informatieverwerking zich nog in fase 2 bevinden, tekent zich duidelijk een nieuwe aanpak af.

3. De huidige inzichten ten aanzien van gegevensverwerking en -opslag gaan uit van de gedachte dat gegevens niet gezien moeten worden vanuit één enkele toepassing of bedrijfsfunctie. Integendeel, gegevens vormen juist de verbinding tussen bedrijfsfuncties, aangezien hun primaire doel is 'het uitwisselen van informatie'. Tevens is er een duidelijke tendens, technieken voor de gegevensverwerking te ontwikkelen, welke de gegevensverwerking vanuit de uiteindelijke gebruiker gezien, zo natuurlijk mogelijk doet plaatsvinden: de gebruikersbehoeften bepalen in sterke mate soort en vorm der toe te passen technieken.

### 8.1.1 Databases

Deze inzichten en tendensen waren aanleiding tot het concipiëren van het begrip '*database*' en de database-technieken.

Een database kan misschien het beste omschreven worden als:

> Een *gemeenschappelijke* verzameling van met elkaar *samenhangende gegevens* en de *relaties* hiertussen, welke *tegelijkertijd* toegankelijk zijn voor verscheidene eindgebruikers.

In deze omschrijving is een aantal essentiële aspecten benadrukt:
- De database is de *gemeenschappelijke* verzameling. D.w.z. de opgeslagen informatie is voor gemeenschappelijk gebruik en dient voor de communicatie tussen gebruikers.

- De database bevat de samenhangende gegevens en hun relaties - de in de database opgeslagen gegevens representeren zoveel mogelijk de 'natuurlijke' informatiepatronen, waarmee de eindgebruikers vertrouwd zijn.
- Verschillende gebruikers kunnen tegelijkertijd toegang verkrijgen tot de database. Aangezien immers de database gemeenschappelijk is voor de communicatie tussen gebruikers, zullen deze ook zo min mogelijk op elkaar moeten wachten, maar juist zoveel mogelijk tegelijk in de database hun informatie moeten kunnen vinden c.q. opslaan.

### 8.1.2 Eigenschappen van database systemen

Met bovenstaande punten zijn eigenlijk de belangrijkste eigenschappen van een database systeem genoemd.
Toch dient ook een aantal additionele eigenschappen genoemd te worden, welke ook bepalend zijn voor de bruikbaarheid van een database:

1. Data organisatie - data onafhankelijkheid
   Een database systeem moet de 'natuurlijke' structuur der informatie op gemakkelijke en flexibele wijze kunnen representeren. Hierbij moeten overbodige duplicatie en aanvullende systeeminformatie voor de gebruiker overbodig zijn.
   De data organisatie mogelijkheden moeten onafhankelijk zijn van het gebruikte computersysteem of de toe te passen programmeertalen.

2. Data manipulatie
   Een database systeem moet een voldoende aantal benaderingsmethoden omvatten, zodat een eindgebruiker op eenvoudige wijze zijn informatie in de database kan onderbrengen en kan terugvinden. Hierbij moet hij de informatie op zo natuurlijk mogelijke wijze kunnen identificeren, c.q. op zo gemakkelijk en efficiënt mogelijke wijze zijn informatiebehoefte kunnen omschrijven.

3. Opslagstructuur - opslag onafhankelijkheid
   Een database systeem moet efficiënte technieken bevatten om de logische structuren op de fysiek beschikbare opslagmedia efficiënt te kunnen vastleggen en benaderen, zonder de logische structuur geweld aan te doen.
   Deze opslag en benadering moeten op eenvoudige wijze aan veranderende benaderingspatronen en frequenties kunnen worden aangepast, zonder dat de gebruiker hiervoor zijn benaderingsmethode hoeft te veranderen.

4. Gemeenschappelijk gebruik
Een database systeem moet het mogelijk maken dat meer dan één gebruiker tegelijkertijd in de database werkzaam kan zijn. Hierbij dient het systeem efficiënte mogelijkheden te bieden om gebruikers te vrijwaren tegen tegenstrijdigheden welke door elkaars aanwezigheid in het systeem kunnen ontstaan.

5. Bescherming (privacy)
Een database systeem moet de opgeslagen informatie beschermen tegen oneigenlijk of ongeautoriseerd gebruik.

6. Betrouwbaarheid
Een database systeem moet garanderen dat ingebrachte informatie onvervormd bewaard blijft en teruggevonden kan worden.
Een database systeem moet de gebruikers voldoende mogelijkheden bieden de logische juistheid der opgeslagen informatie te bewaken en te toetsen.

7. Herstelmogelijkheden
Een database systeem moet voldoende mogelijkheden bieden om de database - of beter de informatie - na het optreden of ontdekken van een fout of storing zo snel mogelijk weer in de juiste staat terug te brengen.

8. Herstructurering
In een database systeem moet de opgeslagen informatie, zowel qua inhoud als qua structurering op betrekkelijk eenvoudige wijze aan veranderende gebruiksomstandigheden kunnen worden aangepast.

9. Beheer
Een database systeem moet een aantal hulpmiddelen bieden om:
- De database eenvoudig te kunnen creëren, onderhouden, reorganiseren.
- Nieuwe gebruiksbehoeften (programma's) te testen zonder de betrouwbaarheid van de database inhoud aan te tasten.
- De database inhoud op beschikbaarheid, betrouwbaarheid en efficiëntie te testen.
- De gebruiksgewoonten en -frequenties te registreren.

10. Uitwisselbaarheid
Een database systeem moet garanderen dat bij vernieuwing of verandering der gebruikte systeemtechnieken en hulpmiddelen, de invloed op bestaande gebruikersprocedures minimaal zijn, c.q. dat deze procedures tenminste nog correct bruikbaar zijn, hoewel deze dan misschien niet de additionele mogelijkheden ten volle benutten.

### 8.1.3 Database Management Systemen

Database systemen als zodanig worden bepaald door de gebruiksomgeving van een database. In totaliteit is een database, dus het database systeem gebruiksgebonden. Deze gebruiksgebondenheid wordt echter bepaald door de informatie-inhoud en -structuur en door de gevolgde informatie-gebruikende procedures van de uiteindelijke gebruikers.
De specifieke opslag- en benaderingstechnieken en de hulpmiddelen zijn veel algemener van aard. Daarom is er, analoog aan de situatie rond de conventionele bestandsorganisaties, een scheiding ontstaan tussen toepassingstechnieken (application software) en database systeemtechnieken (database system software). De programmatuur en technieken voor deze laatste categorie worden meestal aangeduid als: Database management systemen (DBMS).

Een aantal DBMS-pakketten is op de programmatuurmarkt verkrijgbaar, zoals bijvoorbeeld:
TOTAL
PHOLAS
DMS 1100
SYSTEM 2000
IMS
UDS
IDMS

### 8.1.4 Codasyl DBTG

Aangezien vele van deze pakketten onafhankelijk van elkaar ontwikkeld zijn, volgen zij meestal eigen concepties. Maar bij de opkomst der database mogelijkheden zijn er ook duidelijk stromingen ontstaan om algemeen aanvaardbare concepties en technieken te ontwikkelen. Een der groeperingen van gebruikers en fabrikanten, die dit trachten te bereiken, zijn de database subcommittees van de CODASYL organisatie: database task groups DBTG. CODASYL zelf is een overkoepelende organisatie van gebruikers en fabrikanten, die tracht algemene programmeertalen en technieken te definiëren. Deze Conference on Data Systems Languages bestaat sinds 1959 en is het meest bekend door haar eerste grote werk: de formulering van COBOL. Sinds het eind der zestiger jaren houden subcommittees zich bezig met de formulering van database technieken en talen hiervoor.

Een eerste voorstel hiertoe werd neergelegd in een rapport, verschenen in oktober 1969. Dit rapport werd uitgebreid bediscussieerd en een groot aantal wijzigingsvoorstellen werd ingediend.

De verwerking hiervan en verdere uitdieping der ideeën resulteerde in een tweede rapport in april 1971. Dit rapport werd wijd en zijd verbreid. Diverse fabrikanten hebben dan ook DBMS systemen ontwikkeld, gebaseerd op de voorstellen in dit rapport.
Voorbeelden zijn:
DMS 1100 van UNIVAC
PHOLAS van Philips
UDS van Siemens
Inmiddels zijn nieuwe verbeterde voorstellen verschenen in 1975.

Ook andere concepties voor database behandeling worden ontwikkeld. Zo geniet de benaderingswijze 'relational database' tegenwoordig ruime bekendheid. De grondslag voor deze benadering berust op de principes van mathematische relatie theorie en werd als zodanig het eerst voorgesteld door E.F. Codd.

De verdere behandeling van database technieken in dit hoofdstuk is echter gebaseerd op de CODASYL DBTG voorstellen van april 1971.

8.1.5 Database talen

In het algemeen onderscheidt men twee fundamentele categorieën van database management systemen.
De ene categorie wordt gevormd door systeem programmatuur en 'gebruikerstalen' welke een uitbreiding vormen voor de gebruiksmogelijkheden van een bestaande hogere programmeertaal:
De applicatie-programmeur beschrijft de procedures voor informatieverwerking in termen van een 'uitgebreide' programmeertaal. De basistaal wordt dan gastheer taal (host language) genoemd en het database management systeem een gastheer taal systeem (host language system).
De andere categorie database management systemen wordt gevormd door de systemen, die rechtstreeks door de eindgebruiker aanspreekbaar zijn zonder de tussenkomst van computerprogramma's. Deze systemen noemt men wel opzichzelfstaand (selfcontained).

De CODASYL-voorstellen gaan uit van gastheer systemen omdat deze benadering meestal algemener en elementairder kan zijn. De opzichzelfstaande systemen kunnen met behulp van een gastheer systeem gemaakt worden. Algemeen kan gesteld worden, dat de gastheer taal systemen dichter bij het computersysteem staan, de opzichzelfstaande systemen dichter bij de uiteindelijke gebruiker.

De gebruiker (programmeur) hanteert de database management mogelijkheden door 'taaluitdrukkingen':

1. De informatie en informatiestructuur worden gedefinieerd met behulp van een:
   Data definiëringstaal (Data definition language - DDL)
   De taal is zelfstandig en niet afhankelijk van een gastheer taal.
   Zie ook paragraaf 8.2.6.

2. De opslagstructuur wordt gedefinieerd met hetzij hiervoor geschikte elementen uit de DDL (CODASYL DBTG 1971), hetzij door een apart hiervoor ontworpen taal:
   - Device Media Control Language DMCL (voorgesteld door CODASYL DBTG 1971 als concept, maar niet uitgewerkt).
   - Opslagstructuur taal (Storage Structure Language SSL) o.m. door PHOLAS van Philips.

   De taal is zelfstandig en niet afhankelijk van een gastheer taal.
   Zie ook paragraaf 8.3.3.

3. Het opslaan en de verwerking van de informatie wordt bestuurd door toevoeging van geschikte taalelementen aan een gekozen gastheer taal - de data manipulatie talen (data manipulation language - DML).
   Het CODASYL-DBTG voorstel werkt dit concept uit voor de gastheer taal COBOL.
   Zie ook paragraaf 8.5.4.

Vooralsnog zijn geen algemene taalelementen voorgesteld voor additionele hulpmiddelen voor database beheer en herstel. Diverse fabrikanten volgen hier hun eigen inzichten.

### 8.1.6 Voor- en nadelen

Het zal wellicht duidelijk zijn, dat het toepassen van database technieken een aantal voordelen voor de uiteindelijke gebruiker (en de programmeur) oplevert.

- Gecoördineerd beheer van de (gemeenschappelijke) informatie in een gebruikersomgeving en de gemakkelijke bereikbaarheid van deze informatie.
- Het verband tussen informatie-eenheden behoeft niet meer door de gebruiker moeizaam tussen de gegevens gelegd te worden, maar kan ook in de database vastgelegd worden.
- Vermijden van het op verschillende manieren en op verschillende plaatsen vastleggen van dezelfde informatie, zodat ook de problemen van het in overeenstemming houden van deze verschillende representaties vermeden worden.
- Door de fysieke opslagtechnieken niet tot de eindgebruiker te laten doordringen, kan deze zich concentreren (en beperken) tot de voor hem logische en natuurlijke hantering der informatie.

- Eenvoudiger bescherming der informatie tegen ongeoorloofd gebruik en betere controle-mogelijkheden voor juist gebruik.
- Uitgebreider en natuurlijker mogelijkheden om een behoefte aan informatie te formuleren.
- Informatie-opslag onafhankelijk van van te voren reeds bekende informatie gebruiksbehoeften: de informatiestructuur laat ook informatie-gebruik toe, die niet ten tijde van het database ontwerp voorzien werd.

Toch is er ook een aantal nadelen te noemen. Deze nadelen zullen bij gebruik op de koop toe genomen moeten worden, of men moet voldoende maatregelen treffen om ze acceptabel te doen zijn:

- De centralisering der informatie en de uitgebreide mogelijkheden tot informatie-opslag en -verwerking kunnen veroorzaken dat een bedrijf of organisatie in zijn functioneren zeer afhankelijk wordt van de goede werking van een database systeem.
  Problemen met een database kunnen dan de oorzaak zijn dat het bedrijf komt stil te liggen. Hiertegen helpt de verhoging der betrouwbaarheid niet afdoende.
  Alternatieve vervangings- en uitwijkmogelijkheden moeten ook aanwezig zijn.
- Het gemeenschappelijk gebruik der database kan ongewenste invloed uitoefenen op de resultaten van een individuele benadering. Het systeem moet hiervoor de nodige regulerende mogelijkheden bezitten.
- De geïntegreerde informatie in een database is meestal complexer dan de geautomatiseerde informatieverwerking vóór de toepassing der database.
  Hierdoor is het informatieverwerkende systeem lastiger te beheren.

Ook zuiver menselijke factoren kunnen een nadelige invloed hebben:

- De natuurlijke weerstand tegen iets nieuws en vreemds.
- Verlies van (vermeende) eigendomsrechten op opgeslagen informatie.
- Applicatie-programmeurs hebben minder invloed op database ontwerp.
- Door de eenvoudiger programmeer-mogelijkheden vrezen programmeurs overbodig te worden.

De praktijk wijst echter uit dat de belangrijkste winst voor een organisatie bij het gebruik van een database systeem gevonden wordt in de discipline en controle op de informatieverwerking die ontstaat door de inspanning en de organisatie nodig om het systeem op te zetten en te ondersteunen.

## 8.2 *Data organisatie in databases*

### 8.2.1 Entiteitenmodel

Voor de vastlegging van informatie in databases gaat men veelal uit van het zgn. entiteitenmodel:

Men onderkent in een organisatie, waarvoor informatie in een database opgeslagen moet worden, de basiseenheden en begrippen. Bijv. in een verkooporganisatie de klanten, de artikelen, de bestellingen, de afleveringen, de medewerkers, etc.
Dit zijn allemaal personen, tastbare dingen, of reëele zaken, zowel concrete voorwerpen als abstracte begrippen. Ze hebben echter allen gemeen, dat zij - in de betreffende organisatie - als 'reëele feiten des levens' onderkend worden. Deze worden met een algemene term entiteit (of object) aangeduid.
Voor de goede gang van zaken in de organisatie is het nodig om informatie uit te wisselen over deze entiteiten. Hiervoor moeten we de entiteiten kunnen omschrijven. Dit doen we door vastleggen van de eigenschappen (attributen, proporties) van de entiteiten, die voor ons doel belangrijk zijn. Een eigenschap wordt gekenmerkt door twee grootheden:
- eigenschaptype - de naam van de eigenschap
- eigenschapwaarde.

Bijv. Een eigenschap van een persoon is, dat hij een naam heeft. Een andere eigenschap is zijn adres:

| Eigenschaptype | Eigenschapwaarde |
|---|---|
| naam van een persoon | - JANSEN |
| adres van een persoon | - HOOFDSTRAAT 3 |

Als we dus een lijst van bij elkaar horende eigenschappen opsommen, d.w.z. een lijst van eigenschaptypen met bijbehorende waarden, dan duiden we hiermee een entiteit aan. Voorbeeld:

| Eigenschaptype | Eigenschapwaarde |
|---|---|
| naam | JANSEN |
| adres | HOOFDSTRAAT 3 |
| woonplaats | HAARLEM |
| geboortedatum | 15-7-1928 |
| etc. | |

resultaat entiteit 'persoon JANSEN'
(informatie over de man zelf)

Als we alleen de lijst van eigenschaptypen beschouwen, dan kunnen we alleen constateren, dat hiermee een persoon omschreven kan

worden. M.a.w. de lijst van eigenschaptypen kenmerkt het entiteittype.
De lijst van bijbehorende eigenschapwaarden duidt een verschijningsvorm (occurrence) van het entiteittype aan.

Resumerend kan gesteld worden dat entiteiten en eigenschappen als volgt omschreven worden:

| | | |
|---|---|---|
| eigenschap | ⟶ | eigenschaptype, eigenschapwaarde |
| entiteit | ⟶ | entiteittype, entiteit-verschijningsvorm |
| entiteittype | ⟶ | lijst van (relevante) eigenschaptypen |
| entiteit verschijningsvorm | ⟶ | lijst van bij de eigenschappen behorende waarden. |

Alle eigenschappen van een entiteit kunnen gebruikt worden om een entiteit aan te duiden - te identificeren.
In de praktijk zijn echter vaak enkele eigenschappen hiervoor voldoende. Voorbeeld:

| Entiteittype | Eigenschaptype |
|---|---|
| persoon | naam |
| | adres |
| | woonplaats |
| | geboortedatum |
| | salarisnummer |
| | salaris |

'Naam-adres-woonplaats' zal in veel gevallen de entiteit voldoende identificeren, evenals 'salarisnummer'.

Hiervan maken we gebruik om informatie over een entiteit te weten te komen: We noemen de identificatie en vragen dan naar de bijbehorende waarden van andere eigenschaptypen; of - in antwoord op een vraag - we identificeren de entiteit en noemen de bijbehorende waarden van andere eigenschaptypen. Voor de identificatie is het voldoende om een combinatie van eigenschappen te kiezen, die een entiteit binnen de omgeving, die ons interesseert, eenduidig bepaalt.
Bijv.: Landelijk gezien zullen 'naam-adres-woonplaats' en misschien 'geboortedatum' nodig zijn om eenduidigheid te bereiken. Binnen één afdeling in een bedrijf echter kan de 'naam' alleen al voldoende zijn.

Binnen een bedrijf kan 'salarisnummer' voldoende zijn, terwijl binnen een gemeente het salarisnummer in het geheel niet relevant is en dus onbekend. Dit voorbeeld geeft aan dat zelfs 'sleuteleigenschappen' binnen een omgeving binnen een andere omgeving onbelangrijk, niet bruikbaar of onbekend kunnen zijn.
Resumerend: Informatie is de vastlegging of communicatie van een selectie van eigenschappen van een entiteit. Deze selectie wordt bepaald door de omgeving waarin de eigenschappen betekenis hebben, hetzij voor identificering van de entiteit, hetzij voor informatie-uitwisseling over de entiteit, hetzij voor beiden.

Tussen entiteiten kunnen *relaties* onderkend worden.
Bijv.
- een medewerker *behoort* tot een afdeling
- een artikel *ligt opgeslagen in* een magazijn
- een klant *bestelt* een artikel

Dergelijke relaties tussen entiteiten kunnen zelf ook als een entiteit onderkend worden, vooral als aan de relatie relevante eigenschappen toegekend worden. Voorbeeld:

een klant *plaatst een order voor* een artikel

| klant --- | order --- | artikel |
|---|---|---|
| - adres | - aantal | - prijs |
| - korting | - leverdatum | - voorraad |
| | - leverprijs | |

het bestelde aantal, de leverdatum en de (uiteindelijke) leverprijs zijn duidelijke eigenschappen van de relatie 'order' tussen 'klant' en 'artikel'.

Relaties kunnen bestaan tussen:
- één verschijningsvorm van een entiteittype en één verschijningsvorm van een ander entiteittype:

  1 : 1 (1 op 1)

  bijv. een persoon *is geboren in* een geboorteplaats.

  Bij iedere verschijningsvorm van de ene entiteit behoort altijd een verschijningsvorm van de andere entiteit - bij iedere A hoort één B:




- één verschijningsvorm van een entiteitsoort en 0, één of meer verschijningsvormen van een andere entiteit:

  1 : n (1 op n)

bijv. een afdeling b e s t a a t uit 0, 1 of meer medewerkers.

Bij iedere verschijningsvorm van de ene entiteitsoort behoort een aantal (0, 1, ...) verschijningsvormen van de andere entiteitsoort; bij iedere verschijningsvorm van de andere entiteitsoort hoort echter slechts één verschijningsvorm van de ene entiteitsoort - bij een A hoort 0, 1, 2, ... B's; bij iedere B behoort slechts één A.




- diverse verschijningsvormen van de ene en de andere entiteitsoort:




(n op m)

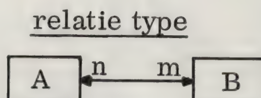
bijv. een magazijn bevat diverse artikelen; maar een artikel kan in meer dan één magazijn worden opgeslagen.

Bij een verschijningsvorm van een entiteitsoort kunnen 0, 1 of meer verschijningsvormen van de andere entiteitsoort behoren en omgekeerd:

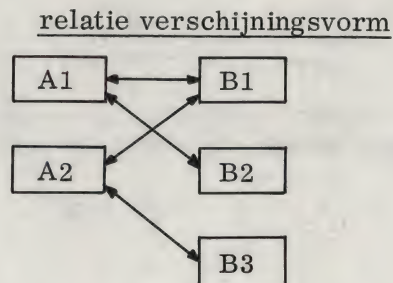



De database conceptie volgens de CODASYL voorstellen vereist echter dat deze laatste relatiesoort niet rechtstreeks in een database gehanteerd wordt. Derhalve dient deze vorm 'vertaald' te worden. Dit kan bijvoorbeeld door de relatie als zodanig als een entiteit te onderkennen (in de praktijk is dit veelal toch al nodig, omdat eigenschappen van de relatie vastgelegd moeten worden). Vervolgens worden relaties onderkend tussen de 'relatie entiteit' en de beide oorspronkelijke entiteiten.
Bijv.:

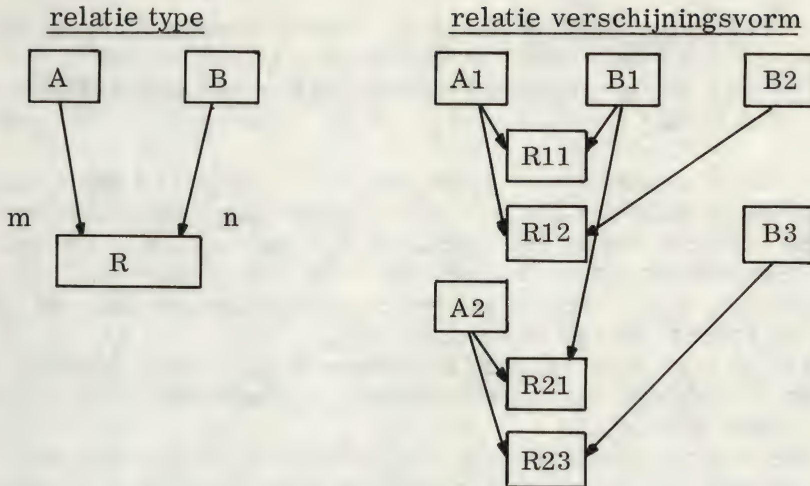
relatie type




relatie verschijningsvorm



wordt vastgelegd als:




Vaak is er geen duidelijk onderscheid tussen een eigenschap van een entiteit en de relatie met een andere entiteit.
Voorbeeld:
Een medewerker woont in een woonplaats.
Eigenlijk is hier sprake van een relatie tussen medewerker en woonplaats. In de praktijk wordt deze relatie echter vaak als een eigenschap 'woonplaats' van de medewerker gehanteerd.

Het omgekeerde is ook mogelijk. Dit komt vooral voor, indien de eigenschap van een entiteit meer dan één waarde tegelijk kan hebben voor één verschijningsvorm van de entiteit.
Voorbeeld:
Het golfbereik van een radio. Een radio kan meer dan één golfbereik hebben. Dit is in eerste instantie een eigenschap. Men kan hem echter ook opvatten als een relatie tussen één radio en één of meer golfbereiken.

8.2.2 Records en data-items

Wanneer wij de CODASYL voorstellen volgen, dan is de kleinste gedefinieerde informatie-eenheid in een database de data-item. Een dergelijk data-item kan dan weliswaar opgebouwd zijn uit één of meer karakters, maar deze karakters afzonderlijk representeren geen informatie in de database. Bijv.:
de naam van de medewerker: JANSEN.
De afzonderlijke letters J, A, N, S, E en N echter vertellen ons niets van de entiteit medewerker Jansen.

In een database is dus de data-item het kleinste ondeelbare element: de elementaire informatie-eenheid.

Eigenschappen van entiteiten en relaties worden in de database vastgelegd met behulp van data-items.
De waarde van de eigenschap wordt dan gerepresenteerd door de inhoud van het data-item.

In de CODASYL concepties wordt een record gedefinieerd als een benoemde verzameling van nul, één of meer data-items. Het ligt dus voor de hand het record te associëren met een entiteit waarvan de eigenschappen dan vastgelegd worden in de data-items van het record.
Meervoudige eigenschappen kunnen dan vastgelegd worden met behulp van vectoren en repeterende groepen.
Een vector is in dit verband gedefinieerd als een ééndimensionale geordende verzameling van verschijningsvormen van een elementair data-item in een record.
Een repeterende groep is een geordende verzameling van verschijningsvormen van een groep data-items, vectoren of repeterende groepen in een record.
Vectoren en repeterende groepen zijn speciale vormen van een groepering van data-items. In CODASYL concepties wordt in het algemeen een groep data-items aangeduid door het begrip data-aggregatie (data-aggregate).

Uit het oogpunt van flexibiliteit en efficiency verdient het echter aanbeveling in de meeste gevallen om meervoudige eigenschappen vast te leggen als (1 : n) relaties, vooral indien de n niet voor alle verschijningsvormen van een entiteittype dezelfde waarde heeft.

Voorbeelden:

1. record met elementaire data-items.

   medewerker:

| Naam | Adres | Woonplaats |
|---|---|---|
| JANSEN | HOFVELD | ARNHEM |
| SMIT | KERKALLEE 3 | VELP |

2. record met vector.

   radio:

| Type | Artikelnummer | Golfbereik (1) | Golfbereik (2) | Golfbereik (3) |
|---|---|---|---|---|
| 5U721 | 912 8731 51312 | MG | LG | |
| 5U731 | 912 8731 51313 | MG | LG | FM |

3. record met repeterende group.

gezinshoofd:

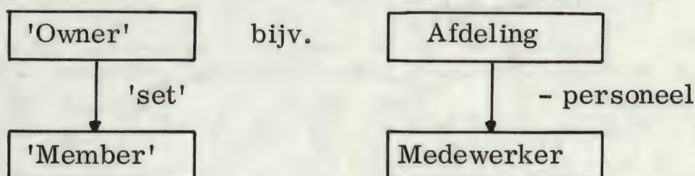
| Naam | kind | | kind | | kind | |
|---|---|---|---|---|---|---|
| | naam | geb. dat. | naam | geb. dat. | naam | geb. dat. |
| KLAASSEN | JAN | 5-3-56 | PIET | 7-8-58 | MARIE | 13-12-60 |

8.2.3 CODASYL sets

Volgens de CODASYL voorstellen kunnen (1 : n) relaties rechtstreeks vastgelegd worden in een database. Voorwaarde is echter, dat deze relaties bestaan tussen verschillende entiteittypen. Hiertoe heeft CODASYL een begrip 'set' gedefinieerd:

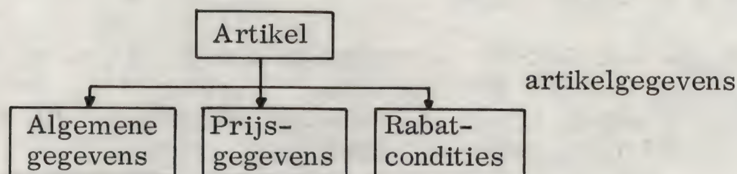
- Een SET (type) is een benoemde verzameling van recordtypen, waartussen een relatie bestaat. Hierbij vormt een recordtype het OWNER type, een of meer andere recordtypen MEMBER typen.
- Een SET verschijningsvorm wordt gevormd door één verschijningsvorm van het OWNER recordtype en 0, 1 of meer verschijningsvormen van de MEMBER recordtypen.
- Een recordtype kan niet tegelijk OWNER type en MEMBER type in één en hetzelfde settype zijn.
- Een recordtype kan OWNER type zijn in 0, 1 of meer settypen en tegelijkertijd MEMBER type in 0, 1 of meer andere settypen.
- Een record verschijningsvorm van het OWNER recordtype impliceert een verschijningsvorm van het settype waarvan het OWNER is.
- Een verschijningsvorm van een MEMBER recordtype participeert slechts dan in een set verschijningsvorm, wanneer daadwerkelijk een relatie gelegd is tussen de record verschijningsvorm en de set verschijningsvorm. Deze relatie kan in het algemeen ook weer verbroken worden, zonder dat de record verschijningsvorm uit de database verwijderd wordt.
- Een verschijningsvorm van een MEMBER recordtype kan slechts in één verschijningsvorm tegelijkertijd van een settype voorkomen.

Een dergelijk settype kan in diagram als volgt weergegeven worden:



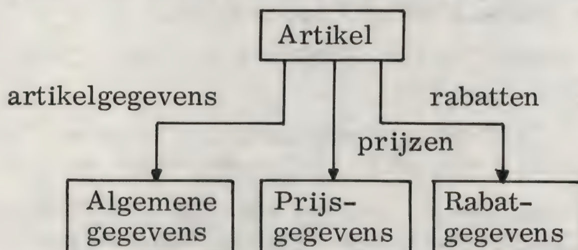
Een set bestaat dus uit een OWNER, een MEMBER, en de relatie ertussen. De pijlpunt indiceert, welk recordtype het MEMBER type is. Een dergelijk diagram wordt wel BACHMAN-diagram genoemd, naar Charles Bachman die deze wijze van set weergave het eerste gebruikte.

Uit de definities blijkt dat een set relatie conceptueel ook meer dan één MEMBER type mag bevatten.
Bijv.:



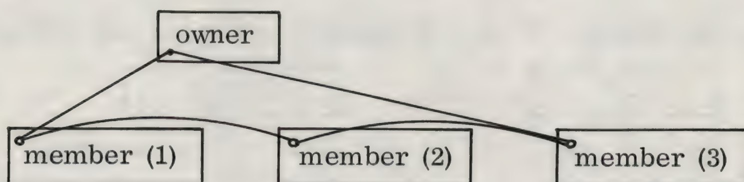

De meeste implementaties beperken zich echter tot één member-type per settype. Dit temeer aangezien het weinig voorkomt, dat één logische relatie bestaat tussen één entiteit en meer dan één andere entiteit.

Bovenstaand voorbeeld kan dan ook zeer goed vastgelegd worden als:

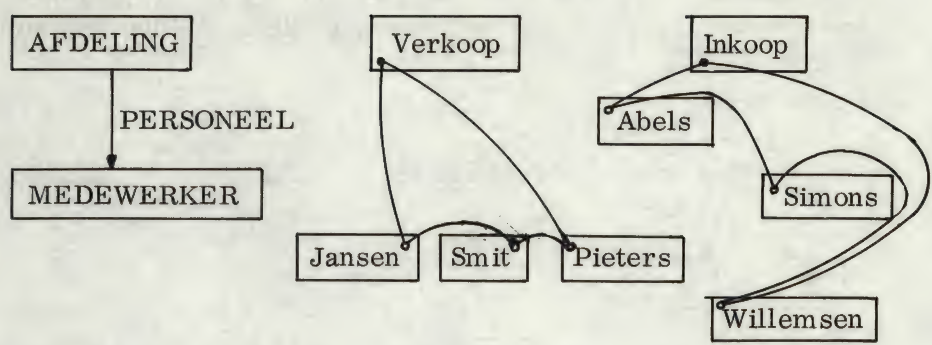



Bovenstaand set-begrip wijkt enigszins af van het algemene begrip uit de wiskunde, etc.
Derhalve wordt (buiten CODASYL geöriënteerde omgevingen) dikwijls de term CODASYL-set of coset gebruikt om dit settype aan te duiden.
De Bachman-notatie wordt gebruikt om settypen en structuren van settypen aan te geven.

Om een set verschijningsvorm uit te beelden, wordt de volgende notatie toegepast:



Voorbeeld:

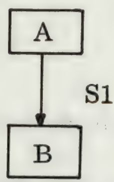



8.2.4 Logische database structuren

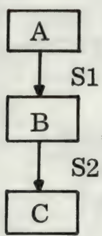
Het geheel van gegevenseenheden (records) en de relaties ertussen, wordt de gegevensstructuur (info-structuur, data-structuur) genoemd. Dergelijke structuren kunnen met behulp van het set-concept vastgelegd worden. Hierbij moet echter wel gelet worden op het onderscheid tussen een structuur van settypen en een structuur van de set-verschijningsvormen. Het samenstel van settypen structuren vormt dan de structuur - voorschrift van de database - het s c h e m a. De set-verschijningsvormen leggen dan de structuur van de informatie vast volgens de regels van het schema.

In de database structuur (schema) kennen wij drie structuur grondvormen:

1. S e q u e n t i ë l e   s t r u c t u u r
   De set grondvorm is een voorbeeld van een sequentieel settype structuur.



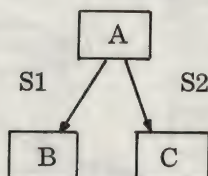

Het kenmerk is dat de recordtypen owner zijn in ten hoogste één settype en eveneens member in één (ander) settype.
De uitgebreider vorm is dan ook:




| Recordtype | Owner | Member |
|---|---|---|
| A | S1 | - |
| B | S2 | S1 |
| C | - | S2 |

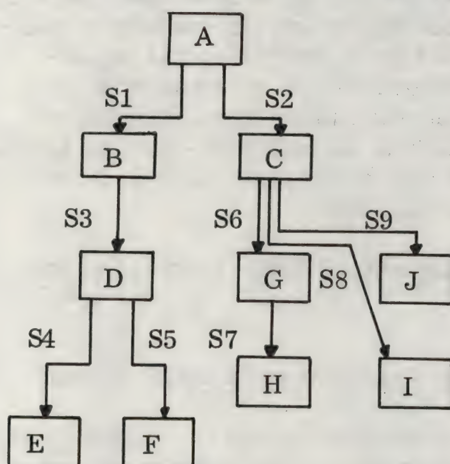
2. Boomstructuur
Het kenmerk van de boomstructuur is, dat een recordtype owner kan zijn van meer dan één settype maar alleen member kan zijn in één settype.

Grondvorm:



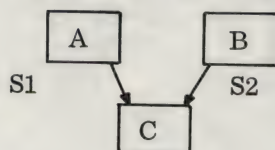

| Recordtype | Owner | Member |
|---|---|---|
| A | S1, S2 | - |
| B | - | S1 |
| C | - | S2 |

Een uitgebreider voorbeeld is het volgende. Hierbij blijkt, enerzijds dat de sequentiële vorm als een speciale vorm van de boomstructuur beschouwd kan worden.
Anderzijds blijkt, dat een structuur een boomstructuur genoemd wordt, indien de structuur tenminste één boomgrondvorm bevat, en verder alleen boom- en sequentiële grondvormen.



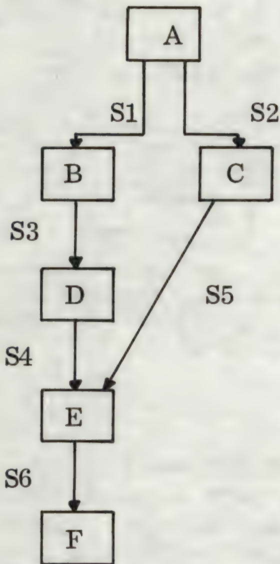

| Recordtype | Owner | Member |
|---|---|---|
| A | S1, S2 | - |
| B | S3 | S1 |
| C | S6, S8, S9 | S2 |
| D | S4, S5 | S3 |
| E | - | S4 |
| F | - | S5 |
| G | S7 | S6 |
| H | - | S7 |
| I | - | S8 |
| J | - | S9 |

3. Netwerkstructuur
Het kenmerk van de netwerkstructuur is, dat een recordtype member kan zijn in meer dan één settype.
Daarnaast kan het echter ook owner zijn in meer dan één settype.
De boomstructuur en de sequentiële structuur zijn dan ook als speciale gevallen van de netwerkstructuur te beschouwen.
De grondvorm van een netwerkstructuur is dus:



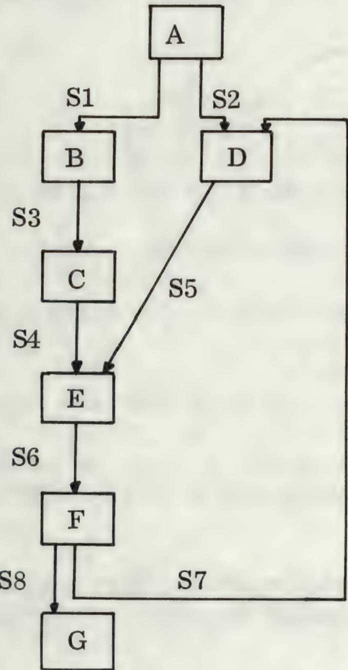

| Recordtype | Owner | Member |
|---|---|---|
| A | S1 | - |
| B | S2 | - |
| C | - | S1, S2 |

Als in een uitgebreider structuur tenminste één keer de netwerk grondvorm voorkomt, dan noemt men de gehele structuur een netwerk.




| Recordtype | Owner | Member |
|---|---|---|
| A | S1, S2 | - |
| B | S3 | S1 |
| C | S5 | S2 |
| D | S4 | S3 |
| E | S6 | S4, S5 |
| F | - | S6 |

Structuren kunnen een cyclisch karakter bezitten, d. w. z. in een dergelijke structuur kunnen één of meer cycli voorkomen. Een cyclus bestaat uit een gesloten sequentie van settypen, zodanig dat de MEMBER van de 'laatste' set de OWNER is van de 'eerste' set.



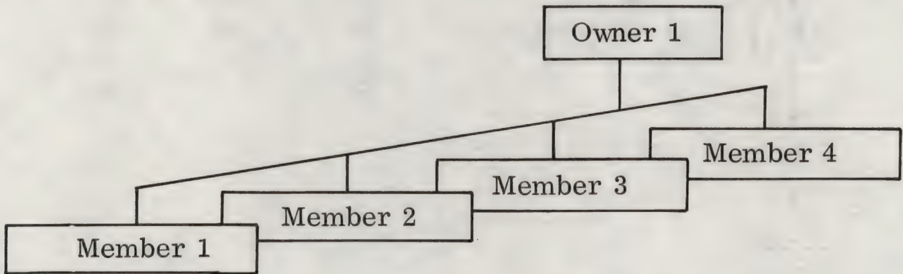

| Recordtype | Owner | Member |
|---|---|---|
| A | S1, S2 | - |
| B | S3 | S1 |
| C | S4 | S3 |
| D | S5 | S2, S7 |
| E | S6 | S4, S5 |
| F | S7, S8 | S6 |
| G | - | S8 |

Cyclus

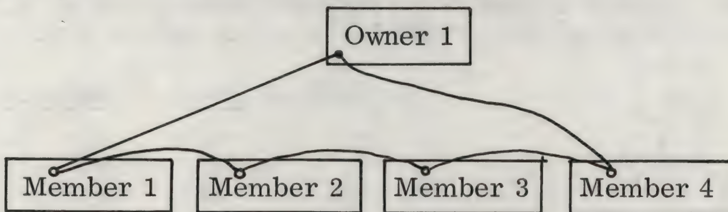
| Recordtype | Owner | Member |
|---|---|---|
| D | S5 | S7 |
| E | S6 | S5 |
| F | S7 | S6 |

In de CODASYL concepties zijn deze cyclische constructies volledig toelaatbaar en verwerkbaar.

De structuur van een set-verschijningsvorm is in principe altijd een boomstructuur. De set relatie is immers 1 : n.
Per set verschijningsvorm heeft één owner een relatie met meer dan één member,




maar ieder member heeft slechts relatie met één owner.
De CODASYL voorstellen staan toe, dat tussen de members onderling een volgorde relatie onderkend kan worden: member 1 wordt altijd direct gevolgd door member 2, etc. CODASYL definieert hierbij zelfs dat member 1 de owner volgt, terwijl het laatste member aan de owner voorafgaat.
In diagram kan dat als volgt weergegeven worden:




Als volgorde criteria kunnen verschillende mogelijkheden gebruikt worden:
- op opklimmende waarde(n) van één of meer data-items, bijv. alfabetisch op naam;
- op positioneringsvolgorde, bijv. altijd achteraan of vooraan nieuwe members toevoegen.

Het volgorde criterium is per settype, dus geldt voor alle verschijningsvormen van een settype.
Indien de volgorde is op opklimmende of afdalende volgorde van één of meer data-items, dan vormen deze items a.h.w. een opklimmende of afdalende sleutel.

CODASYL onderscheidt verschillende relatievormen, gerekend naar de wijze hoe de relatie tussen een member en een set-verschijnings-

vorm tot stand komt. Enerzijds beschouwt men hier het leggen van de relatie, anderzijds de verbreking:

- De relatie tussen een member en een set verschijningsvorm kan automatisch gelegd worden door het DBMS op het moment, dat het member als record in de database wordt gebracht (AUTOMATIC MEMBERSHIP).
  Het alternatief is, dat de gebruiker altijd zelf 'met de hand' de relatie laat aanbrengen tussen een reeds aanwezig record en een set verschijningsvorm (MANUAL MEMBERSHIP).
- De relatie kan per definitie van blijvende aard zijn. D.w.z. de relatie, eenmaal tot stand gebracht, kan slechts verbroken worden door het member uit de database te verwijderen: Een eenmaal gelegde relatie is verplicht (MANDATORY MEMBERSHIP).
  Hier is de tegenstelling, dat een relatie naar eigen keuze weer verbroken kan worden (OPTIONAL MEMBERSHIP).

De combinatie van MANDATORY en AUTOMATIC levert de mogelijkheid, dat een record verschijningsvorm per definitie member moet zijn in een set verschijningsvorm. Dit wordt dan op schema niveau vastgelegd. Vervolgens zal het DBMS er op toezien dat de relaties bij inbreng van een record in de database worden vastgelegd en hierna niet meer worden verbroken. Andere combinaties houden altijd in, dat de gebruiker procedureel de relaties moet leggen en/of verbreken. Hij is daar dan ook verantwoordelijk voor de relaties.

### 8.2.5 Areas

De records en sets, die de informatie vastleggen, worden in de database ondergebracht. Hoewel de fysieke wijze van opslag voor het logisch gebruik niet relevant is, zal toch een zekere controle op de opslag op logisch niveau nodig zijn. Indien bijvoorbeeld een database zo groot is, dat deze niet in zijn geheel actief op de computer aanwezig kan zijn, dan is het noodzakelijk, dat de database logisch in delen onderverdeeld kan worden. Een of meer delen kunnen dan tegelijk actief of non-actief zijn.
Men moet dan echter kunnen aangeven, welke informatie in welk database deel bij elkaar opgeslagen moet worden.
Merk op, dat hierbij niet belangrijk is, hoe het opgeslagen wordt.

Een variant hierop is het archiveren van 'verouderde' informatie. Een apart deel van de database wordt dan bestemd om archiefinformatie te bevatten. Dit gedeelte dient alleen actief te zijn (on-line) als de archiefinformatie nodig is, c.q. indien informatie naar het archief dient te verhuizen.

Een andere mogelijkheid is, dat men tijdens de informatieverwerking behoefte heeft aan een tijdelijk 'klad' deel in de database om hierin informatie tijdelijk op te slaan. De in dit deel aanwezige informatie is dan niet relevant meer zodra het verwerkingsproces beëindigd is.

De mogelijkheid voor deze semi-fysieke opdeling van een database is in de CODASYL-voorstellen, het AREA concept. Hierin wordt de database gezien als een adresseerbare ruimte, onderverdeeld in partities - de AREAS. Ieder record verschijningsvorm wordt in de database opgeslagen op een uniek adres in deze ruimte.
Het geschikte adres wordt gevonden in de AREA, welke ten tijde van inbreng van de informatie in de database wordt bepaald.

Een bijkomend gebruik van het area concept volgens CODASYL is om hiermee toegangsmogelijkheden tot brokken informatie te autoriseren en controleren.
Men dient per area een mogelijke toegang tot de hierin aanwezige informatie van te voren kenbaar te maken. Voor autorisatie kan hieraan een privacy-mechanisme verbonden worden, zodat men het juiste wachtwoord moet meegeven. Voor de controle op gelijktijdig gebruik van de database door meer dan een gebruiker, kan men delen van de informatie, ondergebracht in één en dezelfde area ook per area tijdelijk exclusief reserveren (exclusive access) of beschermen tegen gelijktijdige verandering door anderen (protected access).

8.2.6 Schema DDL

De beschrijving van de recordtypen en settypen, waarin de informatiestructuur vastgelegd wordt in voor een database management systeem bruikbare termen, wordt het S C H E M A van een database genoemd. Hierin ligt dan de l o g i s c h e opbouw van de database vast. De gebruikte taal is de SCHEMA Data Definition Language (DDL).
De beschrijving van de database structuur - het SCHEMA - wordt onderverdeeld in vier groepen:

1. SCHEMA-ENTRY
   Hierin wordt de naam van de database gedefinieerd. Ook kunnen hier autorisatie controles - PRIVACY LOCKS - op het niveau van de gehele database en het Schema hiervoor vastgesteld worden.

2. AREA-ENTRIES
   In deze sectie wordt de opdeling van de database in areas vastgelegd. Ook hier kunnen de autorisatie controles per area gedefinieerd worden.

3. RECORD-ENTRIES
   Vervolgens worden de recordtypen gedefinieerd. Per record worden hier de record karakteristieken vastgelegd en tevens uit welke data-items het recordtype bestaat.
   Autorisatie controles per recordtype kunnen hier gedefinieerd worden.
4. SET-ENTRIES
   De laatste groep bevat de definities voor de settypen en met welke recordtypen als OWNERS en MEMBERS zij de relaties vormen.
   Autorisatie controles op set niveau kunnen ook vastgesteld worden.

In de record-entries en set-entries kunnen per recordtype of settype sleutels (search keys) gedefinieerd worden. De records kunnen met behulp hiervan sneller geïdentificeerd worden.
Zie ook paragraaf 8.5 - Database manipulatie.

## 8.3 *Fysieke opslag van databases*

### 8.3.1 Opslag onafhankelijkheid

Een van de belangrijke eigenschappen van een goed database management systeem is, dat de fysieke opslag van de data gescheiden wordt van de logische representatie. Hiermee wordt bereikt dat de gebruiker zich niet hoeft te verdiepen in de wijze van fysieke opslag, noch in de optimalisering hiervan, als hij de gegevens logisch verwerkt. De fysieke kenmerken mogen dan ook niet doordringen in het logische interface met de gebruiker.
Dit houdt niet in, dat de gebruiker de fysieke opslag niet zou kunnen bepalen of beïnvloeden. Natuurlijk dient de fysieke opslag ook van buitenaf te definiëren en te besturen te zijn. De daarbij betrokken karakteristieken en grootheden zijn onder meer:
- wijze van administratie der opgeslagen record verschijningsvormen;
- fysieke vastlegging der set relaties met bijbehorende administratie;
- aantallen opgeslagen of te verwachten record verschijningsvormen en set relaties;
- de te gebruiken achtergrondgeheugens met welke adresseringstechnieken;
- lengte en blokkingsfactor van de fysieke records.

Niet alleen de aantallen records, etc. bepalen de optimale keuze van deze grootheden, maar ook de te gebruiken benaderingsmethoden in de database en de frequenties hiervan. Ook de dynamiek van de informatie - de vervangingsgraad en de informatiegroei - beïnvloeden de optimale dimensionering van de fysieke opslag.

Het is heel goed mogelijk, dat in de loop van de tijd dat de informatie in de database gebruikt wordt, bepaalde gebruikspatronen zich wijzigen. Het is dan zeer belangrijk de fysieke opslag naar de veranderde situatie te richten zonder dat dit de constructie van de gebruikersprogramma's doet veranderen. Een gebruiker mag dit alleen merken, doordat gemiddeld genomen zijn database benadering sneller verloopt.

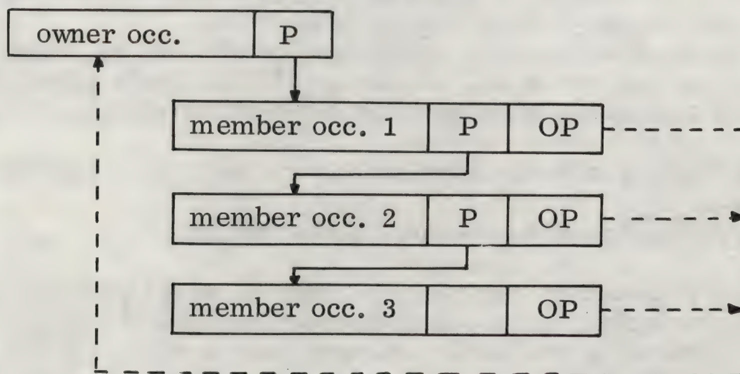
8.3.2 Opslagstructuren

De opslagstructuur wordt in twee niveaus verdeeld:
- de fysieke database structuur
- de device opslag.

Het eerste niveau is de fysieke afbeelding van de database. Hierin zijn alle logische elementen terug te vinden in hun fysieke gedaante. De basiseenheid blijft het logische record, echter eventueel aangevuld met interne database administratie, welke niet zichtbaar is voor de gebruiker.
In een database heeft iedere logische record verschijningsvorm altijd minstens één unieke identificatie - de database key. Deze heeft een door de DBMS gegenereerde unieke interne waarde. De gebruiker beschikt echter over middelen deze waarde als identificatie te gebruiken.

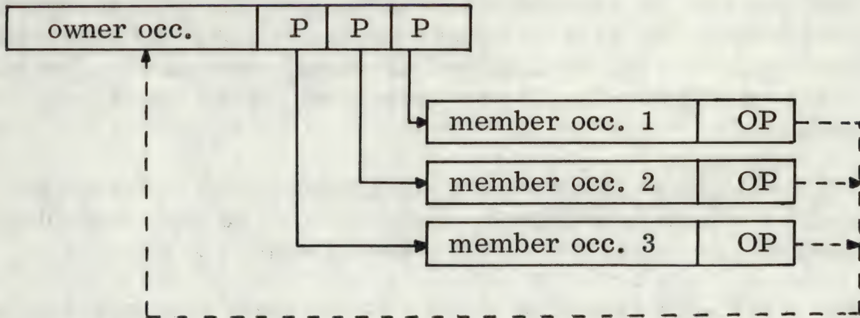
Voor het aangeven van de relatie tussen OWNER's en MEMBER's in de diverse SET's kan een aantal technieken gebruikt worden zoals kettingadressering (chaining), verwijsadressen (pointer array) en fysiek sequentiële verwijzingen.

Kettingadressering (chaining)




P = POINTER
OP = OWNER POINTER
occ. = occurrence (verschijningsvorm)

Verwijsadressen (pointer array)




De verwijsadressen worden in de praktijk meestal in een gesorteerde tabel opgeslagen, die via verwijzing met de owner verbonden is. Indien de set volgorde d.m.v. een opklimmende of afdalende sleutel wordt gedefinieerd, dan wordt deze (primaire) sleutel ook in de tabel opgenomen.

Fysiek sequentieel

| | | | |
|---|---|---|---|
| owner occ. | member occ. 1 | owner occ. 2 | member occ. 3 |

Een SET verschijningsvorm wordt in één of meer fysieke blokken opgeborgen.
Meestal bevat ieder member verschijningsvorm ook een terugverwijzing (OP) naar de owner waar het bij hoort.
Welke van deze technieken gekozen wordt, hangt voor een groot deel af van de soort applicatie en de mogelijkheden die het 'database management' systeem geeft. De keuze van de te gebruiken techniek heeft een grote invloed op de verwerkingsefficiency.

Op set relaties kunnen ook secundaire sleutels gedefinieerd worden (search keys). Deze worden dan in hulptabellen vastgelegd. Fysiek zijn deze hulptabellen òf in sequentiële volgorde van secundaire sleutels gevuld, òf d.m.v. een randomising techniek.

Het tweede niveau van de fysieke opslag is de werkelijke opslag op de achtergrondgeheugens. De basiseenheid hierin is het fysieke record of blok. De database inhoud - de gebruikersgegevens èn de interne administratie - wordt over de blokken gedistribueerd en zo op de geheugenmedia opgeslagen. In de meeste gevallen worden hiervoor magneetschijven gebruikt. De daarbij meest gebruikte opslagvorm is de relatieve bestandsorganisatievorm.

Logische records (verschijningsvormen) worden, al dan niet aangevuld met interne administratieve gegevens, geblokt in fysieke blokken. Meestal bevatten de fysieke blokken een of meer gehele logische records. Soms echter komt ook 'spanning' voor, d.w.z. een logisch record wordt wegens zijn grootte verdeeld over twee of meer fysieke blokken.

Adrestabellen of 'randomising' technieken leggen verband tussen de logische record identificatie (sleutels, etc.) en de fysieke blokken waar de record verschijningsvormen zijn opgeslagen.

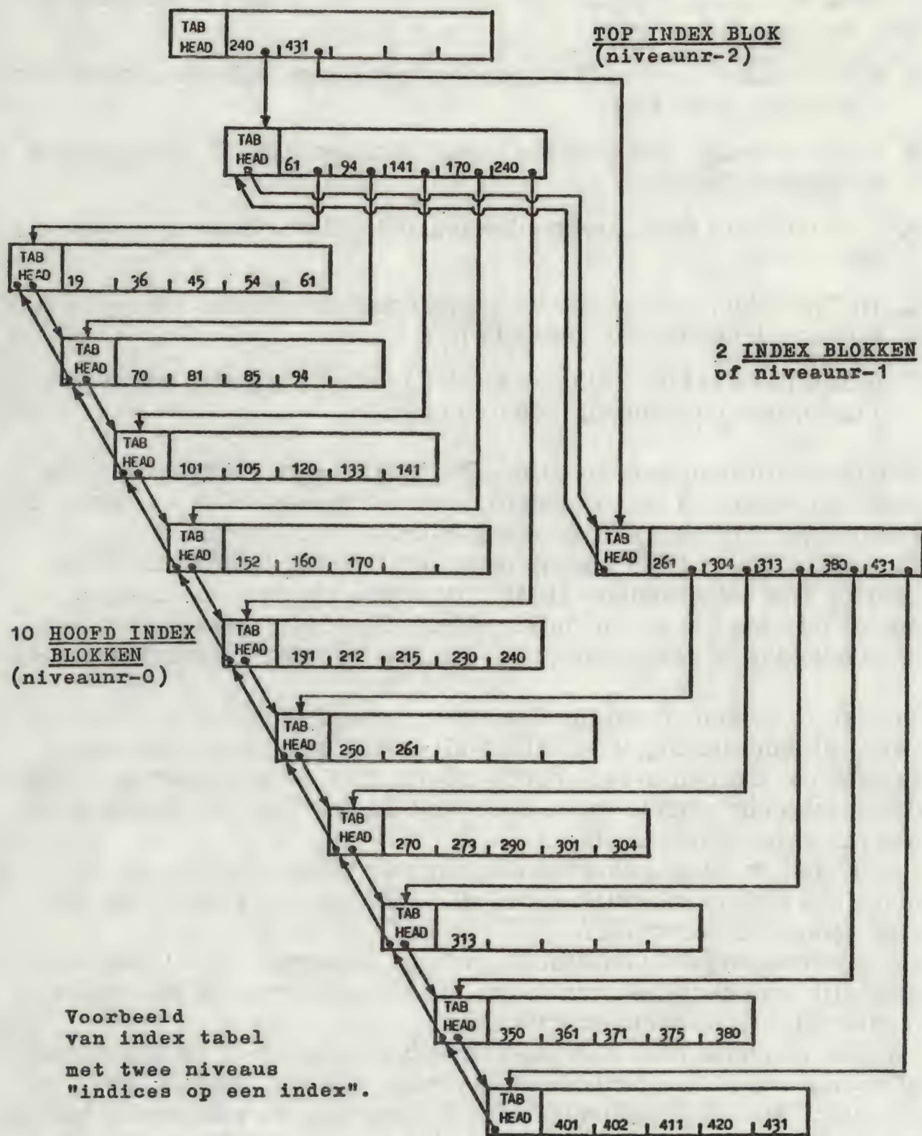
De hulptabellen voor sleutels en andere interne administratie worden ook 'geblokt' in de fysieke blokken opgeslagen. Indien de tabellen gesorteerd zijn (pointer array, indexed search key), dan wordt vaak intern een indexeringstechniek op deze tabellen toegepast zodra een tabel over meer dan één fysiek blok verdeeld is. Deze index op de index bevat dan per entry de hoogste key waarde, die in een bepaald blok voorkomt plus de verwijzing naar dat blok. Dit kan op meer dan één niveau plaatsvinden (index op de index van de index). (Zie fig. 28 op blz. 129.)

### 8.3.3 DDL-elementen of SSL voor opslagdefinitie

CODASYL heeft voor de fysieke opslagstructuur definitie een aantal taalelementen aan de DDL toegevoegd. Deze elementen definiëren o.m.:
- een 'randomising' relatie tussen een sleutel en de plaats in de database;
- de implementatievorm van een set relatie in pointer array, chain, etc.

Voor de definiëring van de opslag grootheden geeft CODASYL geen taalelementen, doch volstaat met de suggestie, dat een DBMS fabrikant dit zelf doet in een DMCL (device media control language).

De Philips en Siemens implementaties van de CODASYL voorstellen gaan een stap verder. Deze hebben alle fysieke definiëringen ondergebracht in een aparte taal, de Storage Structure Language SSL. Hieraan zijn ook kwantificeringselementen toegevoegd. Deze definieren ook gedeeltelijk de opslag grootheden. De overige opslag parameters worden vastgelegd d.m.v. JCL in de creatiefase van de database.


TOP INDEX BLOK
(niveaunr-2)
TAB HEAD 240 431
TAB HEAD 61 94 141 170 240
2 INDEX BLOKKEN
of niveaunr-1
TAB HEAD 19 36 45 54 61
TAB HEAD 70 81 85 94
TAB HEAD 101 105 120 133 141
TAB HEAD 152 160 170
TAB HEAD 261 304 313 380 431
10 HOOFD INDEX BLOKKEN
(niveaunr-0)
TAB HEAD 191 212 215 230 240
TAB HEAD 250 261
TAB HEAD 270 273 290 301 304
TAB HEAD 313
TAB HEAD 350 361 371 375 380
TAB HEAD 401 402 411 420 431
Voorbeeld
van index tabel
met twee niveaus
"indices op een index".

Figuur 28.

## 8.4 *Data betrouwbaarheid en bescherming*

Bij de opslag en verwerking van informatie speelt het begrip 'betrouwbaarheid' (reliability) een grote rol. Deze betrouwbaarheid kent een aantal aspecten:

1. Geheimhouding. Zekerheden tegen ongeoorloofd gebruik der informatie (privacy).
2. Veiligheid. Zekerheden tegen verloren gaan der opgeslagen informatie (security).
3. Accuratesse. Zekerheden tegen onjuistheid der gegevens (accuracy).
4. Integriteit. Zekerheden tegen conflicten tussen de opgeslagen informatie-eenheden (integrity).
5. Consistentie. Zekerheden dat juiste informatie ook fysiek juist wordt opgeslagen (consistency).

Van deze vijf punten heeft alleen het laatste punt betrekking op de juiste werking van het gebruikte database management systeem. Dit kan dan ook door het DBMS zelf gecontroleerd worden.
De andere vier punten hebben betrekking op het juiste beheer en gebruik van het gebruikte DBMS. Hier kan het systeem hooguit mogelijkheden bieden om juist gebruik te helpen controleren door de beheerder of gebruiker (bijv. input-screening, privacy checks).

Vooral na de komst van de databases, waardoor een aantal gebruikers, al dan niet tegelijkertijd, vrij gemakkelijk toegang kunnen krijgen tot allerlei gerelateerde informatie, is het duidelijk geworden (vaak door schade en schande) dat de betrouwbaarheidscontrole een primaire noodzaak is.
Ook is aan het licht gekomen dat controle alleen niet genoeg is, maar dat het ook mogelijk moet zijn, om een geconstateerde fout snel te kunnen herstellen.
De beschikbaarheid (availibility) van de informatie moet dus zo kort mogelijk onderbroken worden om het systeem weer in betrouwbare toestand terug te brengen (recovery).
Om dit te kunnen bereiken moeten in het systeem en de informatie al voorbereidende maatregelen genomen worden, voordat een fout optreedt. Deze kunnen ondermeer bestaan uit het regelmatig trekken van een copie van de informatie op een gedefinieerd moment dat de informatie nog betrouwbaar is (back-up), het administreren van de wijzigingen, die plaatsvinden (logging) en het meer dan eens opnemen van vitale informatie (controlled redundancy).

8.4.1 Geheimhouding (privacy)

Onder bescherming van gegevens, 'geheimhouding', verstaan we het voorkomen van ongeautoriseerde toegang tot die gegevens. Zo'n bescherming moeten we kunnen bieden op verschillende niveaus. We zullen gegevens moeten kunnen beschermen tegen lezen, wijzigen en verwijderen. Ook moeten we in staat zijn te voorkomen, dat relaties tussen bepaalde gegevens kunnen worden vastgelegd of verbroken. Een voorbeeld kan dit verduidelijken.

Figuur 29 geeft de gegevensstructuur weer van een stukje personeelsadministratie: per medewerker zijn o.a. geregistreerd de salarisgegevens en de medische gegevens.
Het zal duidelijk zijn, dat niet iedereen toegang mag hebben (inzage mag hebben) in de salarisgegevens van een medewerker. In nog sterkere mate geldt dit wellicht voor zijn medische gegevens. Deze gegevens zullen we dus moeten beschermen tegen lezen en wel zodanig dat de salarisgegevens alleen toegankelijk worden gemaakt voor functionarissen en/of programma's van de salarisadministratie, en de medische gegevens uitsluitend en alleen voor de bedrijfsarts en zijn staf.

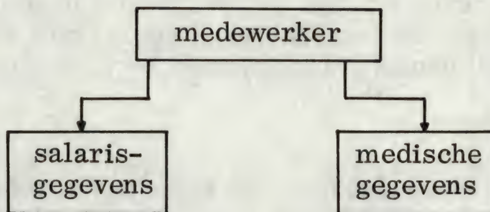



Fig. 29

Ten aanzien van het wijzigen van salarisgegevens zullen waarschijnlijk verdergaande maatregelen moeten worden getroffen dan voor lezen, terwijl een nog grotere beperking moet bestaan voor het verwijderen van de salarisgegevens. Wijzigingen kunnen bijvoorbeeld worden voorbehouden aan de personeelchef.
Het kan ook voorkomen, dat t.b.v. de vaststelling van begrotingen of ter wille van de statistiek de salarisgegevens of de medische gegevens moeten kunnen worden gelezen, maar dan los van de personen op wie gegevens betrekking hebben. Dan moet dus bescherming worden geboden tegen het vinden van relaties met deze personen.

Merk op, dat de bescherming gericht is tegen ongeoorloofde toegang van personen tot de gegevens.
Bijv. een persoon mag geen kennis nemen van de gegevens.

Een database handler programma zal dus wel een LEES-opdracht mogen uitvoeren, als het programma de ingelezen informatie maar niet uitprint; de informatie mag ook niet zichtbaar worden d.m.v. een door de gebruiker opgeroepen dump van het computergeheugen.
Er bestaan verschillende technieken om bescherming te realiseren. De vier meest voorkomende zullen we hier kort bespreken.

a. Sleutels

De geautoriseerde gebruiker(s) van bepaalde gegevens wordt een sleutel (Engels: privacy key) verschaft met behulp waarvan het slot (Engels: privacy lock) dat op die gegevens is gelegd, kan worden geopend. Zoals in een niet-geautomatiseerde omgeving bepaalde geclassificeerde gegevens worden verstrekt na het tonen van een pasje dat slechts na 'screening' wordt uitgegeven, zo zullen gegevens uit een (geautomatiseerde) gegevensverzameling pas worden verstrekt nadat een wachtwoord (Engels: password) in orde is bevonden.
Als met bestanden wordt gewerkt, zouden dergelijke wachtwoorden in de bestandlabels kunnen zijn opgenomen. In een geïntegreerde gegevensverzameling (database) kunnen dergelijke wachtwoorden zelfs worden geregistreerd op record-niveau en op veldniveau. De gebruiker van de gegevens moet vooraf zijn wachtwoorden opgeven als hij bepaalde beschermde bestanden of gegevens wil verwerken.

b. Separatie

Separatie (fysieke scheiding) van gegevens wordt bewerkstelligd door ze op te slaan in kluizen of althans zodanig dat ze fysiek moeilijk bereikbaar zijn.
Als van bestanden t.b.v. computerverwerking gebruik wordt gemaakt, kan deze methode bijv. worden gerealiseerd door leesbare instructies op haspel (Engels: tape reel) of schijvenpakket (Engels: disk packet) te zetten.
De beslissing over het plaatsen van een dergelijke haspel of pakket op de apparatuur moet dan worden genomen door de bestandsbeheerder.
In een database is een dergelijke bescherming mogelijk door vitale gegevens in aparte areas onder te brengen die alleen na speciale opdracht op de computer opgebracht worden.

c. Codering

Zoals sommige geclassificeerde informatie kan worden vastgelegd in een bepaald geheimschrift, waarbij de lezer (gebruiker) dan een codeersleutel nodig heeft om die informatie te ontcijferen, kunnen in bestanden en databases ook bepaalde gegevens in een gecodeerde vorm worden vastgelegd. De programma's zullen

dan een procedure moeten definiëren die deze code omzet in een bruikbare vorm.

d. Incomplete beschrijving
Door gedrukte of geschreven informatie slechts te verstrekken aan uitdrukkelijk belanghebbenden (bijv. met behulp van een verzendlijst) kan eveneens in zekere mate bescherming worden geboden. In een gautomatiseerd systeem kan dit worden gerealiseerd door een recordtype in het programma zodanig te beschrijven, dat alleen de relevante gegevens zijn gedefinieerd. Meestal zal dan wel de lengte van de 'weggelaten' gegevens moeten worden opgegeven ('filler').
In een database is het bovendien mogelijk om door middel van een subschema slechts die gegevens te vermelden, die voor een specifiek probleem relevant zijn.

Belangen zullen moeten worden afgewogen door een onpartijdige instantie, die verantwoordelijk moet zijn voor de uitgifte van wachtwoorden aan de gebruikers van die bestanden. Deze instantie heeft dan tot taak te bemiddelen tussen de eventuele gebruikers van de gegevens en de naleving van de door hem uitgevaardigde voorschriften te controleren.
Naarmate de bestanden meer zijn geïntegreerd in alles omvattende gegevensverzamelingen (databases), zal deze functie belangrijker worden. Een database is immers in principe toegankelijk voor iedereen.

Tenslotte zij nog vermeld, dat bescherming van gegevens niet betekent, dat de programmeur minder zorgvuldig behoeft te zijn als hij die gegevens gaat verwerken. Integendeel, hij zal zich zeer bewust moeten zijn van de consequenties van een verandering die hij aanbrengt in beschermde gegevens.
Een extreem voorbeeld. Als iemand die daartoe gerechtigd is, een medewerker verwijdert uit een gegevensverzameling met de structuur van figuur 29, dan dient hij er ook voor te zorgen dat de betreffende salarisgegevens en medische gegevens worden verwijderd. Anders zou het nog kunnen voorkomen dat iemand anders alsnog de salarissen kan achterhalen van medewerkers die reeds zijn vertrokken (aangenomen is dat met de verwijdering van de medewerker ook het slot op de informatie verwijderd is). Overigens is ook één van de taken van de informatiebeheerder de controle op het voorkomen van dergelijke situaties.

### 8.4.2 Veiligheid (security)

De veiligheidsaspecten hebben betrekking op de fysieke veiligheid van het computersysteem en de achtergrondgeheugens waarop de informatie is opgeslagen.

Dit zijn dus beveiligingen tegen:
- brand
- diefstal
- vernietiging
- beschadiging
- sabotage

etc.

In deze beveiliging wordt meestal voorzien door het vaststellen van menselijke procedures en gedragsregels (alleen beveiliging tegen brand kan min of meer automatisch geschieden).
Goede gedragsregels en een goede controle op de naleving hiervan zijn dus een eerste vereiste.

8.4.3 Accuratesse (accuracy)

Voor een gebruiker van gegevens is het natuurlijk van belang dat de gegevens inderdaad juist zijn. Helaas kan een informatiesysteem hiervoor nooit instaan, de verantwoordelijkheid voor de correctheid der gegevens is een volledige gebruikersaangelegenheid.
Wèl is het mogelijk om met behulp van het informatiesysteem de juistheid enigermate te controleren, bijv. door gebruik te maken van controleprogramma's, etc.

Bijvoorbeeld in een personeelsadministratie systeem kan de regel gelden, dat iemand uit een bepaalde vakgroep een salaris moet hebben, dat tussen zekere grenzen ligt.
Een controleprogramma kan nagaan of aan deze eis voldaan wordt. Een afwijking duidt op een fout. Het feit echter, dat het salaris van een bepaalde medewerker binnen de grenzen ligt, garandeert nog niet, dat het vastgelegde salarisbedrag (in de data) inderdaad juist is.

8.4.4 Integriteit (integrity)

Het begrip integriteit der gegevens behelst, dat de opgeslagen gegevens volledig zijn en elkaar niet tegenspreken.
Voorbeeld: in een informatiesysteem voor bank-giro verkeer geldt de eis, dat een giro-betaling bestaat uit een afboeking bij de betalende rekeninghouder en een bijboeking bij de ontvangende rekeninghouder. Het totaal saldo van alle rekeningen blijft hierbij hetzelfde. Is aan deze eis voldaan, dan is de informatie integer - zelfs als het bedrag der giro-betaling onjuist zou zijn.
Dit laatste is een inbreuk op de accuraatheid.
Ook hier geldt dat slechts de gebruiker (de mensen) kan bepalen of de gegevens aan integriteitseisen voldoen. Een systeem immers kan de gegevens niet b e g r i j p e n.

Deze integriteitscontroles kunnen in applicatie routines geprogrammeerd worden: deze routines vergelijken logische informatie met in de database ingebrachte controle informatie. De applicatie routines kunnen de tegenspraak in de informatie dus wel ontdekken.
De mogelijkheid bestaat om delen van het database management systeem te gebruiken om de integriteit te bewaken. Zo kan een DBMS systeem ervoor zorgdragen, dat bij bepaalde handelingen (bijv. opslag van een bepaald recordtype) applicatie gedefinieerde controle routines uitgevoerd worden.

8.4.5 Consistentie (consistency)

Een opslag van gegevens wordt consistent genoemd, als deze technisch foutloos is.

Voorbeelden:
- De index entries in een index-sequentieel bestand refereren naar de juiste cylinders en sporen.
- Records in een sequentieel bestand staan inderdaad in opklimmende volgorde van een zoeksleutel.
- De 'forward-pointer' in een kettingstructuur refereert inderdaad aan het record, waarin de 'backward-pointer' aan het record, waarin de 'forward-pointer' stond, terug refereert.

De bewaking der consistentie kan geheel door het DBMS uitgevoerd worden, aangezien voor deze controle het niet nodig is, dat de logische interpretatie der gegevens wordt begrepen.

Een database, die niet consistent is, is per definitie ook niet als integer of accuraat aan te merken. Men weet immers niet, of schijnbaar juiste informatie te vertrouwen is. Evenzo is een niet integer database niet accuraat, ook al lijkt de informatie juist.

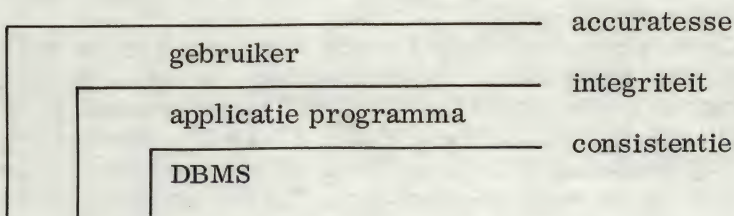



Fig. 30 Werkingssfeer der graden van betrouwbaarheid.

## 8.5 *Database manipulatie*

### 8.5.1 Subschema concept en DDL

Onder database manipulatie worden alle handelingen verstaan, die gegevens in een database opslaan, of gegevens uit de database opzoeken en verkrijgen.
Aangezien een database dient om de gezamenlijke gegevens, welke binnen een organisatie van belang zijn, samen te brengen, zal meestal een gebruiker niet in alle gegevens tegelijk geïnteresseerd zijn. In het algemeen geldt, dat bij ieder gericht gebruik van de database slechts een deel van de database, een subset, relevant is.

Om nu een dergelijke subset te kunnen aangeven, heeft CODASYL het begrip s u b s c h e m a geïntroduceerd:

> Een subschema benoemt in termen van areas, records, data items, en sets, de subset van de database waarop een gebruiker (applicatie-programma) opereert.

Merk op, dat een subschema dus niet de records, sets, etc. definieert. Wel kan de gebruiker in het subschema aangeven, in welke vorm hij de record gegevens aan hem gepresenteerd wenst. Hij kan dus de record-lay-out voor zijn programma aanpassen.
Niet alleen kunnen bepaalde data-items weggelaten worden, ook volgorde wijzigingen van de data-items zijn mogelijk. Hierbij verandert vanzelfsprekend de record-lay-out in de database niet.

Dit concept van een separaat schema en subschema is belangrijk vanuit vele gezichtspunten:
- Zoals reeds genoemd, behoeft de gebruiker (en de applicatie-programmeur) zich niet bezig te houden met de gecompliceerheid van de volledige database, maar kan hij zich beperken tot de voor hem relevante gedeelten.
- Doordat de gebruiker alleen toegang kan hebben tot de in het subschema benoemde subset van de database, wordt de geheimhouding en integriteit van de rest van de database automatisch gewaarborgd.
- Veranderingen in de database zullen alleen dan veranderingen in een programma kunnen veroorzaken als de veranderingen wijzigingen in de subset teweeg brengen. Via de subschema's kan dan de koppeling met de betrokken programma's gelegd worden.

De subschema's leggen voor de programma's de database subset vast. Er is dus een nauwe relatie tussen een applicatieprogramma en een subschema. Merk echter op dat verschillende programma's gebruik kunnen maken van hetzelfde subschema. Wegens de nauwe relatie tussen programma en subschema is voor de beschrijving van

een subschema een taal - de subschema DDL, gekozen, die qua syntax past bij de taal waarin een programma geprogrammeerd is, de gastheer taal (host language).
CODASYL heeft dit reeds uitgewerkt voor de gastheer taal COBOL. Voor andere gastheer talen (bijv. FORTRAN) is men dit aan het voorbereiden.

8.5.2 Manipulatie functies

Ook in de database kunnen de manipulatievormen ondergebracht worden in de groepen:
- inbrengen van gegevens
- opzoeken van gegevens
- kennis nemen van gegevens
- veranderen van gegevens
- verwijderen van gegevens.

Bij al deze manipulaties worden de gegevens benaderd: het access, hetzij om de gegevens op te zoeken en te verwerken, hetzij om een plaats op te zoeken om de gegevens in onder te brengen.

In databases moet duidelijk onderscheid gemaakt worden tussen logisch access en fysiek access.
Het fysiek access zal bijna altijd bestaan uit een direct access met behulp van het relatieve adres van het fysieke blok. Een logisch access wordt dan herleid tot één of meer fysieke accessen van blokken om het gezochte logische record te vinden en eventueel hiertoe database administratie te raadplegen.
Het logisch access is de access-vorm waarmee de gebruiker (de applicatie-programmeur) te maken heeft. Alle hierbij betrokken fysieke accessen en de administratie raadpleging worden voor hem door het DBMS verricht.
De verschillende vormen van logisch access hangen samen met de methoden waarmee records in een database geïdentificeerd kunnen worden. Ook hier maakt men onderscheid tussen sequentieel en direct access, beide vormen zijn echter geheel logisch, d.w.z. hebben geen directe relatie tot de wijze waarop de gegevens fysiek opgeslagen zijn.

Sequentiële access vormen
In deze gevallen wordt het record geselecteerd door zijn logische positie t.o.v. het record dat er voor geselecteerd werd: 'volgend' (next) of 'voorgaand' (prior).
Deze selectie vindt dus plaats in een gedefinieerde volgorde. De volgorde kan dan gedefinieerd zijn binnen de groep van alle records van een bepaald type of binnen gedefinieerde groepen (sets).
Als speciale vormen van sequentieel access worden ook gerekend:

- selecteer de 'eerste' (first)
- selecteer de 'laatste' (last)
- selecteer de tweede, derde, etc. van de groep.

Directe access vormen
Hieronder valt in de eerste plaats de selectie op (een gedeelte van de) inhoud (content retrieval).
De records kunnen algemeen binnen de database gezocht worden, maar ook bestaat de mogelijkheid de zoek naar een record te beperken tot een gespecificeerde groep: het record wordt binnen de van te voren geselecteerde set gezocht. Hierbij wordt dan gebruik gemaakt van de relaties die tussen records kunnen bestaan.
Wanneer bepaalde gedeelten van een record veelvuldig gebruikt (zullen) worden als selectie kenmerk, dan kan men aan deze gedeelten een sneller werkend access-mechanisme verbinden, men definieert deze gedeelten als sleutel in het schema.
Het DBMS zal dan hiervoor zoekmechanismen voorbereiden en deze bij access gebruiken.
Twee zoekmechanismen zijn fundamenteel mogelijk:
- selectie met behulp van tabellen,
- selectie met behulp van plaatsberekening.

Beide mechanieken leggen de relatie tussen de sleutelwaarde en de plaats van het record in de database. Merk op dat deze sleuteltechniek de 'selectie op inhoud' niet vervangt, doch aanvult.
Ook kan het record gedefinieerd worden met behulp van een intern volgnummer, waarmee het database management systeem iedere record verschijningsvorm uniek kenmerkt (database key).

Bovenstaande vormen van identificatie worden toegepast, wanneer een record aan de database wordt toegevoegd of wanneer een relatie gelegd wordt tussen een reeds in de database aanwezig record en een set. Vanzelfsprekend worden ze ook gebruikt bij het terugvinden van een record in de database.
Andere manipulaties op records, zoals vervangen, verwijderen, opereren altijd op het record, dat als laatste door de gebruiker gehanteerd is. Dit record wordt dan het 'huidige' (current) record genoemd. Het DBMS administreert voor iedere gebruiker welk record zijn 'huidige' record is.

### 8.5.3 Gemeenschappelijk gebruik

Een van de kenmerken van een database is, dat verschillende gebruikers op hetzelfde moment toegang kunnen hebben tot de database (concurrent use). Het is dan ook de taak van een DBMS om te zorgen dat deze gebruikers zo weinig mogelijk hinder van elkaar ondervinden. Dit is vrij eenvoudig indien de gebruikers niet dezelfde informatie nodig hebben, of als zij de informatie alleen raadplegen.

Indien echter een gebruiker de informatie verandert, dan kunnen conflicten ontstaan.

Een CODASYL DBMS is er verantwoordelijk voor dat de consistentie van een database intact blijft gedurende een database access. Hierbij is een database access gedefinieerd als één manipulatie (raadpleging, toevoeging, vervanging, verwijdering, etc.).
Voor een afgeronde verandering van informatie in een database kan echter meer dan één manipulatie nodig zijn. Neem het in 8.4 reeds gebruikte voorbeeld van een giro-overboeking. De gehele afgeronde transactie bestaat uit:

1. check het saldo van de betalende rekeninghouder;
2. ga na of de geadresseerde een rekening heeft;
3. boek het te betalen bedrag af van de rekening van de betalende rekeninghouder;
4. boek het te betalen bedrag bij op de rekening van de geadresseerde rekeninghouder.

Iedere stap houdt tenminste één database manipulatie in. Gaat men nu uit van een integere database, dan zal aan het eind der transactie de database weer integer dienen te zijn. Tussen stap 3 en 4 echter is de database niet integer.
Indien de som der rekening saldi op dat moment wordt bepaald, dan zal immers het over te maken bedrag een tekort opleveren.

Alleen de gegevens die in de transactie een rol spelen bepalen de integriteit van de database - althans voor zover het deze transactie betreft. De overige gegevens zijn per definitie integer te beschouwen, omdat zij niet aan de mutatie deelnemen.
Wanneer nu de te muteren gegevens onbereikbaar gehouden worden voor andere gebruikers vanaf het begin der mutatie totdat de transactie gereed is, dan zal een andere gebruiker alleen integere informatie in de database kunnen aantreffen. Tracht een andere gebruiker gedurende de transactie de te muteren gegevens te benaderen, dan dient het DBMS hem te laten wachten totdat de transactie gereed is.
Dit concept houdt dus in dat een transactie de database van een integere staat naar een volgende integere staat brengt. Hierbij geeft de gebruiker (het programma) aan waar de transactie begint en eindigt. Dit brengt met zich mee, dat, indien gedurende de transactie geconstateerd wordt dat een volgende integere staat niet bereikt kan worden, de transactie ongedaan moet worden gemaakt.
D.w.z. dat alle door de transactie reeds gemuteerde gegevens weer hun oorspronkelijke waarde moeten krijgen.

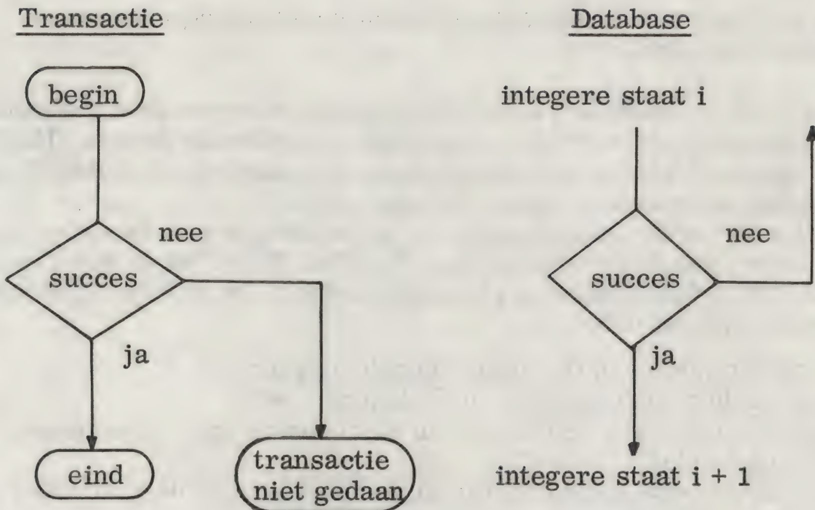



Fig. 31 Werkingssfeer van een transactie.

Het begrip transactie is hier gedefinieerd als:
- de handelingen welke de informatie van een integere staat naar een volgende integere staat muteren, òf - bij onmogelijkheid deze te bereiken - de reeds gemuteerde informatie weer terugbrengt in de integere begintoestand.

De gebruiker kan dus tijdens een gebruik van de database één of meer afgeronde transacties verrichten. Daarom wordt voor dit transactie begrip ook wel de term 'integriteitseenheid' gehanteerd. Het woord transactie wordt dan geassocieerd met het aantal samenhangende integriteitseenheden.

Voorbeeld:
Bij de orderverwerking bestaat één bestelling uit een aantal posten: één post per artikel. De registratie van de verkoop van één artikel, één post, kan dan een integriteitseenheid vormen. De verwerking van de gehele bestelling is dan een transactie. Deze aanpak houdt in, dat per post besloten kan worden of het artikel geleverd wordt. Wordt per volledige bestelling beslist of deze uitgeleverd wordt, dan maakt men van de gehele bestelling één integriteitseenheid.

Deze aanpak maakt het mogelijk dat iedere transactie voor voltooiing ongedaan gemaakt kan worden zonder andere gebruikers nadelig te beïnvloeden. Is de transactie echter gereed en dus de veranderende informatie voor gebruik vrijgegeven, dan kan de verandering alleen ongedaan gemaakt worden door een echte tegentransactie. Hierbij moeten eventueel andere gebruikers wel verwittigd worden.

Ook indien een niet voltooide transactie door storingen in het systeem niet tot een goed einde kan komen, dan kan hij zonder meer ongedaan gemaakt worden.
Dit geldt dus ook voor alle transacties op een bepaald moment: automatisch herstel (automatic recovery) levert dus geen afstemmingsprobleem op, omdat niemand gebruik gemaakt kan hebben van informatie die door een ander in een niet-voltooide transactie gemuteerd is (zie ook par. 8.5.6).
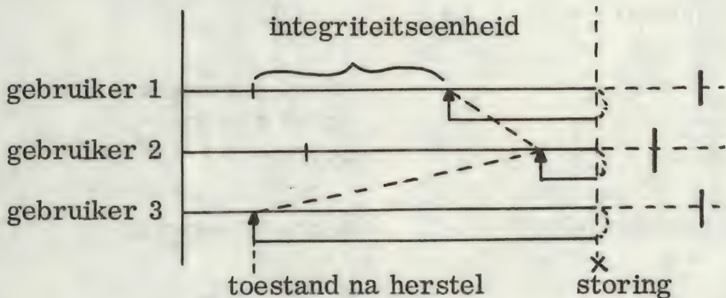



Fig. 32 Automatisch herstel bij systeemstoring.

Voor deze werkwijze dient een te gebruiken DBMS een aantal voorzieningen te bieden.

1. Functies om begin en eind van een transactie aan te geven.
2. Gemuteerde gegevens automatisch onbereikbaar maken voor andere gebruikers, totdat de transactie beëindigd is.
3. Functies om informatie vóór verandering expliciet onbereikbaar te maken voor andere gebruikers, totdat deze weer vrijgegeven wordt of de transactie is beëindigd.
4. Een registratie faciliteit (logging) van de informatie voordat deze gemuteerd wordt ('before image' - nodig om de oorspronkelijke staat te kunnen herstellen). (Zie ook 8.6.4.)

Gegevens onbereikbaar maken voor anderen, zolang men bezig is met gegevens, is dus een noodzaak om anderen te vrijwaren van 'misinformatie'. De ander dient dan te wachten totdat men klaar is. Deze methode biedt echter een kans op wederzijdse uitsluiting of dodelijke omarming (deadlock). Dit houdt in, dat bijv. gebruiker 1 moet wachten op gegevens waarmee gebruiker 2 bezig is, maar zelf gebruiker 2 laat wachten op andere gegevens, waarmee hij zelf bezig is.

Bij een dergelijke dodelijke omarming kunnen ook veel meer dan twee gebruikers betrokken zijn. Een goed DBMS moet een dergelijke

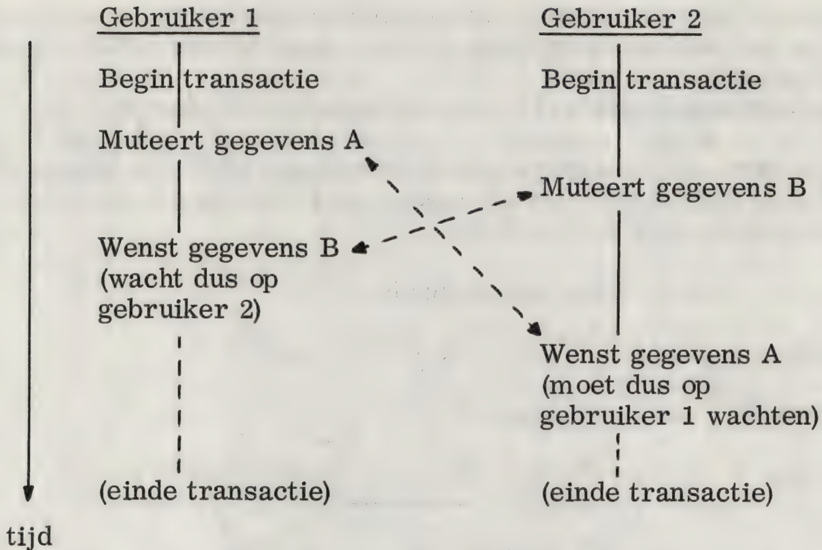



Fig. 33 Dodelijke omarming.

situatie kunnen detecteren of voorkomen. In geval van detectie moet de transactie van een der gebruikers (bijv. degene die als laatste de kring sluit) ongedaan gemaakt worden. Dan kunnen de andere gebruikers voortgaan en de gebruiker, die onderbroken is, kan zijn transactie opnieuw beginnen.

Een eenvoudiger mechaniek om te voorkomen dat gebruikers elkaar ongewild 'misinformatie' kunnen doorgeven, is een gedeelte van de database onbereikbaar te maken voor andere gebruikers, zolang men bezig is. Hierbij maakt men dus ook informatie, die men niet zelf nodig heeft, tijdelijk onbereikbaar.
De methode bestaat uit het exclusief toegang vragen tot een of meer areas van de database.
Een tussenvorm is, wèl gebruikers tegelijk toe te laten, mits zij niet de informatie wensen te muteren.
Deze toegangsvorm wordt door CODASYL 'beschermd' (protected) genoemd.

8.5.4 D.M.L.

Het manipuleren van gegevens in een applicatie-programma wordt eveneens door middel van eenvoudige 'statements' beschreven, de zgn. Data Manipulation Language (DML).
De DML 'statements' zijn symbolische, logische begrippen die alleen refereren naar de data-structuur. De opslagstructuur is hierbij niet van belang.

De beschrijving van gegevens in een bepaalde data- en opslagstructuur via DDL en de manipulatie van deze gegevens via DML, zijn twee volkomen gescheiden methoden, waarmee men bereiken wil dat de applicatie volkomen input/output onafhankelijk wordt.

DML is geen taal in zichzelf, maar maakt gebruik van een gastheer (host) taal en past in het raamwerk van die taal. Alle verzoeken om gegevens ter beschikking te krijgen, toe te voegen, te verwijderen, etc. worden in DML statements geschreven, die als tussenschakel (interface) dienen tussen het programma en het DBMS, die de werkelijke I/O opdrachten uitvoert.
In het bronprogramma (source program) vinden we een mixture van 'host' statements en DML statements, men kan dus stellen dat de programmeur slechts één taal hanteert.

CODASYL heeft voor de Data Manipulation Languages één taal voorstel reeds uitgewerkt. Dit voorstel is voor de gastheer taal (host language) COBOL.
Het oorspronkelijke voorstel dateert van 1971. In 1975 is een gewijzigd voorstel hiervoor gepubliceerd, welke ook ter standaardisering in ANS-COBOL is voorgedragen.
Fundamenteel verschillen de voorstellen niet zoveel, hoewel de formaten soms nogal afwijken.

De belangrijkste functies in de voorstellen zijn:

| CODASYL 71 | CODASYL 75 | |
|---|---|---|
| OPEN | READY | : vraagt toegang tot het deel van de database waarin informatie gemanipuleerd moet worden; definieert het begin van een transactie. |
| CLOSE | FINISH | : beëindigt de verkregen toegang en de transactie. |
| FIND | FIND | : geeft een opdracht aan het DBMS om een record verschijningsvorm te selecteren. |
| GET | GET | : stelt de inhoud van een record verschijningsvorm beschikbaar aan het programma. |
| MODIFY | MODIFY | : het veranderen van gegevens in een van te voren geselecteerd record. |
| STORE | STORE | : het inbrengen van een record in de database. |

| | | |
|---|---|---|
| DELETE | ERASE | : het verwijderen van een record uit de database. |
| INSERT | CONNECT | : het inbrengen van een van te voren geselecteerd record in een set relatie. |
| REMOVE | DISCONNECT | : het verwijderen van een van te voren geselecteerd record uit een set relatie. |
| MOVE | ACCEPT | : het verkrijgen van waarden uit specifieke DBMS velden. |

8.5.5 Database Handler principes

Bij een database worden de standaard benaderingsfuncties voor de applicatie-processen verzorgd door een database handler (DBH). Dit is een van de belangrijkste programma-componenten van een DBMS. Het regelt alle verkeer naar en van de database.

Onderstaande figuur geeft een indicatie hoe een DBH samenwerkt met de applicatie-programma's, het operating system, etc.
De genummerde pijlen geven aan hoe een aanvraag van programma 1 behandeld wordt.

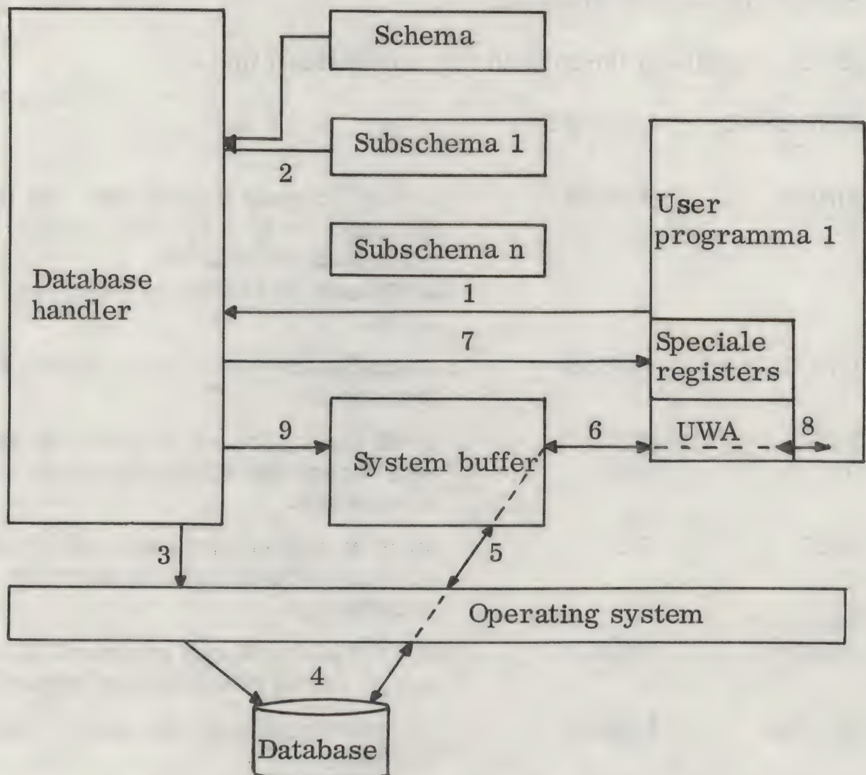



Fig. 34 Schema van een database benadering.

1. Het programma 1 plaatst een aanvraag bij de Data Base Handler (DBH). Alle aanvragen worden in DML statements gedaan.
2. De DBH analyseert de aanvraag en vult de aanvraag aan met informatie uit het schema en subschema voor het betreffende programma. Het schema en subschema zijn oorspronkelijk in DDL statements geschreven.
De DBH maakt gebruik van hieruit gecompileerde object versies.
3. De DBH verzoekt het Operating System om de benodigde fysieke I/O operaties uit te voeren.
4. Het Operating System communiceert met het achtergrondgeheugen (secondary storage).
5. De data worden door het Operating System opgeslagen in systeembuffers.
6. De DBH brengt de gevraagde data van de systeembuffers over naar het gebruikersprogramma in het gebruikerswerkgebied (UWA = User Working Area).
7. Status informatie zoals foutindicaties, area naam en record naam worden door de DBH ook aan het gebruikersprogramma ter beschikking gesteld in speciale registers.
8. Het programma kan nu met de gegevens manipuleren.
9. Hetgeen in de systeembuffers staat, wordt door de DBH bijgehouden, aangezien de mogelijkheid bestaat dat ook andere programma's van dezelfde gegevens gebruik willen maken.

Een van de belangrijke eisen, gesteld aan een database system is, dat verschillende applicatie-processen de database gelijktijdig kunnen benaderen. Een van de functies van de database handler is dan ook, deze gelijktijdige benaderingen zodanig te reguleren en controleren, dat de applicatie-processen elkaar d.m.v. wijzigingsacties in de database niet nadelig beïnvloeden.

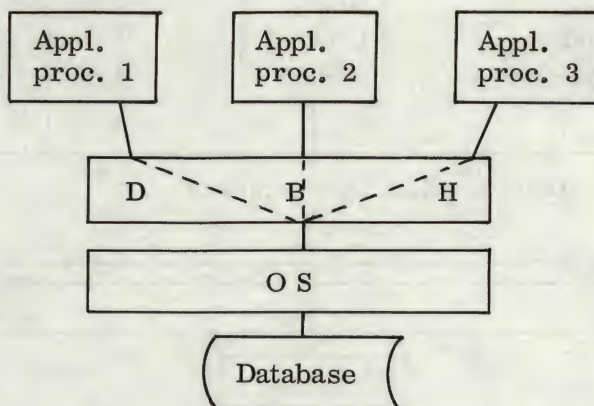
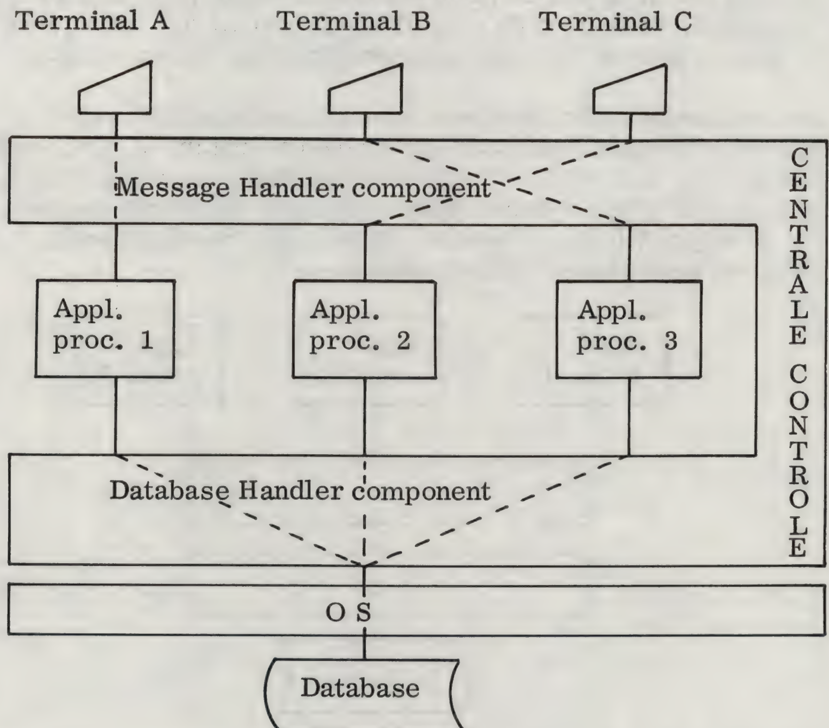



Fig. 35 Schema van een DBH systeem met meerdere gebruikers.

Om bij gelijktijdig gebruik de mogelijkheden, die de database handler biedt, zo goed mogelijk te hanteren, dienen de applicatie-processen in het algemeen 'transaction oriented' opgezet te worden.
Dit past echter volledig in de moderne opvatting, dat processen zoveel mogelijk een afbeelding moeten zijn van de logische verwerking der gegevens zoals de gebruiker zich die voorstelt.

Een database handler reguleert en controleert alleen de benaderingen van een database. Bij real-time verwerking van gegevens, - en zeker wanneer dit plaatsvindt met behulp van decentraal opgestelde terminals - is het ook nodig het berichtenverkeer tussen gebruiker (terminal) en het centrale systeem te regelen.
Hiervoor wordt dan een standaard programma component, de Message Handler toegepast.
Deze ziet er op toe, dat een terminal of invoerstation met het juiste applicatie-proces wordt verbonden.
Maakt het applicatie-proces gebruik van een database, dan blijkt dat vooral de controlerende functies van database handler en message handler veel met elkaar te maken hebben. In feite immers wordt een gebruiker via terminal en applicatie-proces met de database (de gegevens) verbonden.
Om een en ander nu zo optimaal mogelijk te verwezenlijken, is men er toe overgegaan de message handling en database handling te integreren: DB/DC processing.



Met een dergelijk systeem kunnen ook allerlei centrale beheersfuncties ten behoeve van de opgeslagen gegevens op eenvoudige wijze verwezenlijkt worden.
De systeembeheerder kan dan de beschikking hebben over een terminal van waaruit hij het gehele systeem beheert.
Ook bestaat de mogelijkheid op het niveau van de applicatie-processen centrale controles automatisch te laten verrichten zonder dat een individuele terminal gebruiker dit kan vermijden.

## 8.6 *Database beheer*

Bij een geïntegreerd informatiesysteem, welke gebruik maakt van een database, is het van belang dat het systeem aan zijn doel blijft beantwoorden.
Dit kan vanuit twee gezichtspunten benaderd worden:
- de beschikbaarheid van de juiste informatie voor de juiste gebruikers;
- de mogelijkheid met behulp van het geautomatiseerde systeem de informatie efficiënt te kunnen hanteren.

Het eerste aspect heeft dus vooral te maken met de informatie zelf en de (logische) relaties tussen de informatie-eenheden. Het tweede aspect richt zich vooral op de juiste werking van het database management systeem en de technische verwezenlijking van de database.
Om deze redenen onderscheidt men tegenwoordig dan ook meer en meer twee beheersfuncties, te weten:
- data beheer of informatie beheer
- database administratie.

### 8.6.1 Data beheer

De data beheersfunctie beheerst de feitelijke inhoud van een database. Dit houdt onder meer in:

- Vaststellen van de feitelijke inhoud van de database. Hierbij is het van eminent belang, dat de data duidelijk en eenduidig vastgelegd is.
  Standaardizering van data-omschrijvingen spelen dan ook een belangrijke rol.

- Beheer van de betrouwbaarheidsregels en toezien dat deze nageleefd worden.

De data beheersfunctie heeft als het ware het gezag van een eigenaar van de database en vertegenwoordigt de leiding van de organisatie in de database aangelegenheden.

8.6.2 Database administratie

De database administratie functie is verantwoordelijk voor de goede werking van het database systeem.
Dit houdt onder andere in dat:
- de informatie zo efficiënt mogelijk in de database is opgeslagen, en zo nodig gereorganiseerd wordt;
- de database beschikbaar is en technisch toegankelijk;
- de database, na het optreden van een defect, weer zo snel mogelijk hersteld en beschikbaar is;
- statistische gebruiksgegevens verzameld worden en in het algemeen gebruiksefficiëntie gemeten wordt;
- bijstand verleend wordt aan gebruikers en programmeurs voor juiste benaderingswijzen der database;
- toegezien wordt dat alleen uitgeteste en goedgekeurde programma's voor database benadering gebruikt worden.

De database administratie vormt als het ware de technische dienst voor een database systeem.

8.6.3 Hulpmiddelen voor database beheer

Een database management systeem dient ook een aantal hulpmiddelen te omvatten voor de uitoefening der beide beheersfuncties.
Een data beheerder zal vooral behoefte hebben aan goede administratieve hulpmiddelen om de informatie eenduidig te kunnen omschrijven en de gebruiksregels vast te leggen. Hiervoor zijn tegenwoordig zgn. data dictionary/directory systemen beschikbaar, die (in het ideale geval) direct met een DBMS geïntegreerd kunnen worden.
Een ander belangrijk hulpmiddel zijn procedures om het gebruik van de informatie te autoriseren.

Voor de database administrateur (DBA) zijn reeds meer hulpmiddelen voor handen. Hieronder vallen:
- procedures voor herstel van databases;
- procedures om integriteit en consistentie van databases na te gaan;
- procedures om copieën van databases te kunnen maken;
- procedures om het gebruik te meten;
- procedures om (grote) delen van de database versneld te vullen of te legen;
- procedures om de structuur van de database te kunnen rapporteren;
- procedures om een database te kunnen reorganiseren en/of te herstructureren.
  Reorganisatie beslaat de verandering van de fysieke opslag van de database. Herstructurering houdt in dat de logische structuur van de database verandert.

Ook allerlei testfaciliteiten ten behoeve van de database, het DBMS en de gebruikersprogramma's vallen onder de hulpmiddelen voor een juiste database administratie.

Vanzelfsprekend valt het beheer der diverse compilers voor DDL en DML talen binnen het gebied. Hierbij kan opgemerkt worden, dat een goed hulpmiddel om de relatie tussen subschema's en de applicatie-programma's die daarop werken, te registreren, dikwijls node gemist wordt.

### 8.6.4 Database hersteltechnieken

In de inleiding is al opgemerkt, dat het gebied van een database met zich meebrengt, dat een organisatie nogal afhankelijk kan worden van de goede werking van het database systeem.
Het is daarom zeer belangrijk veel aandacht te schenken aan het voorkomen van fouten en storingen.
Toch zal het optreden van fouten en storingen nooit geheel voorkomen kunnen worden. Hierbij zal vaak de inhoud van de database defect raken of in ieder geval niet betrouwbaar meer zijn.

Om een dergelijke defecte database snel te kunnen herstellen is een eerste vereiste, dat hierop al van te voren geanticipeerd is:

- Van tijd tot tijd moet vastgelegd worden dat de database in een betrouwbare staat verkeert (check pointing).
- Uitgaande van een dergelijke situatie moeten alle veranderingen geregistreerd worden die op de database plaatsvinden (logging).
- Op geschikte momenten moet een betrouwbare copie van de database getrokken worden (back-up).

Bij een defecte database zijn in principe twee herstelstrategieën mogelijk.

1. Uitgaande van de defecte database wordt met behulp van de geregistreerde veranderingen de laatste betrouwbare staat gereconstrueerd. Dit is een 'achterwaartse' reconstructie (back-out).

2. Uitgaande van een betrouwbare copie, wordt met behulp van de geregistreerde veranderingen de laatst bereikte betrouwbare staat opnieuw gegenereerd. Dit is een 'voorwaartse' reconstructie (roll forward).

Voor de achterwaartse reconstructie dient van te veranderen gegevens, de toestand vóór de verandering bewaard te worden. Een dergelijke afbeelding noemt men een 'before-image'.
Uitsluitend de eerste before-image na de vastlegging van een

betrouwbare toestand is nodig, daar deze de gegevensstaat van het moment van deze betrouwbare toestand weergeeft, welke voor reconstructie nodig is.

Voor de voorwaartse reconstructie heeft men juist de veranderde toestand nodig. Na iedere verandering wordt de veranderde informatie gelogd. Dit wordt een 'after-image' genoemd. Bij de reconstructie heeft men juist de laatste after-image vóór een vastgelegde betrouwbare toestand nodig, om deze betrouwbare toestand te kunnen reconstrueren.

Beschouw de figuren 37 tot en met 40.
In een applicatie-proces wordt een verschijningsvorm van een record R1 tijdens opeenvolgende integriteitseenheden gewijzigd.
Tussen de eerste wijziging en het daaropvolgende integriteitspunt IP2 moet een after-image R11 worden gemaakt om gebruikt te kunnen

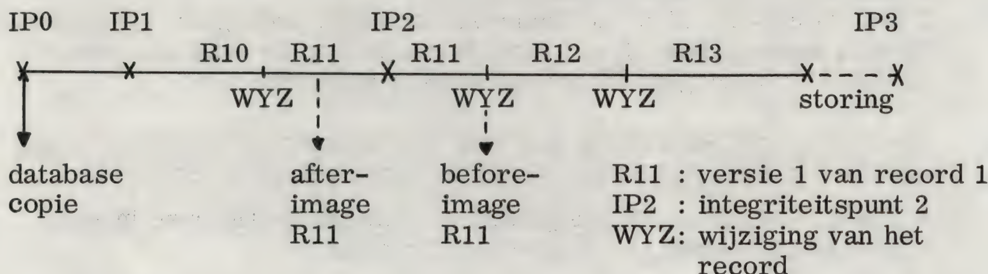



Fig. 37 Een applicatie-proces met logging.

worden voor de voorwaartse reconstructie naar IP2 zoals in figuur 38 is aangegeven.

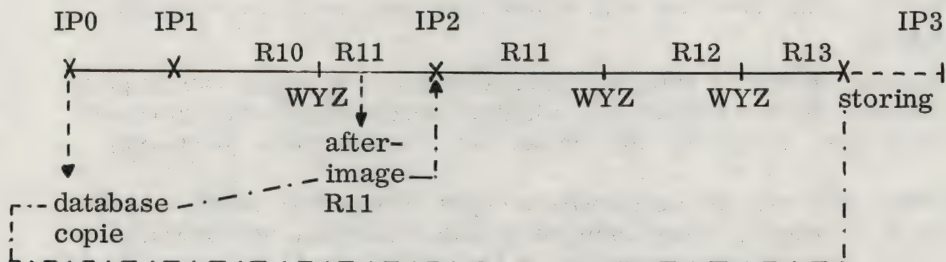



Fig. 38 Voorwaartse reconstructie naar IP2.

Juist voor de tweede wijziging moet er een before-image van R11 worden gemaakt. Deze versie, die geheel gelijk is aan het vorige after-image is nodig voor een achterwaartse reconstructie naar IP2 zoals in figuur 39 is aangegeven.

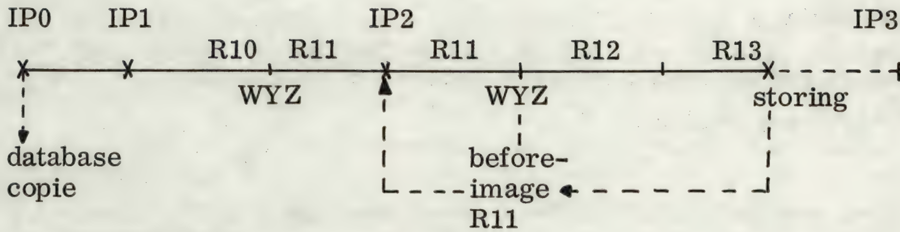



Fig. 39 Achterwaartse reconstructie naar IP2.

In bijna alle gevallen dat er een storing optreedt kan de achterwaartse reconstructie methode gebruikt worden. Alleen als het opslag medium beschadigd is en de inconsistente (of niet integere) database kan niet meer gelezen worden, kan er alleen van een database copie worden uitgegaan om met behulp van de voorwaartse reconstructie methode te worden hersteld. Nadat de database gereconstrueerd is, kunnen de onderbroken transacties opnieuw begonnen worden.

Het concept van de integriteitseenheden houdt in, dat voor ieder proces geldt, dat het laatste integriteitspunt als reconstructiepunt kan dienen zonder in synchronisatie moeilijkheden te vervallen. De lopende transacties immers hebben geen informatie kunnen hanteren die door een andere lopende transactie wordt gemodificeerd.

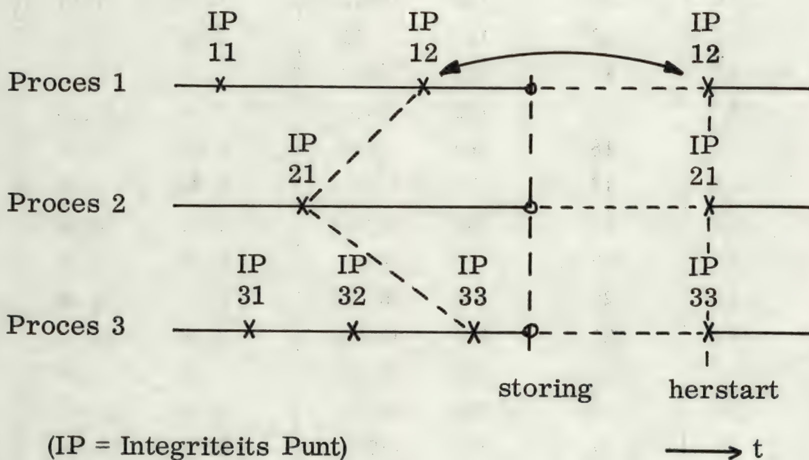



Fig. 40 Automatisch herstel.

APPENDIX 1.

Tabel: Capaciteiten en overdrachtsnelheden voor blokken z o n d e r key (= sleutel).

| octaden per blok | | blokken per | | | overdrachtsnelheid in msec per blok | |
|---|---|---|---|---|---|---|
| minimum | maximum | spoor | cylinder | module | minimum | maximum |
| 1741 | 3625 | 1 | 10 | 2000 | 11.16 | 23.24 |
| 1132 | 1740 | 2 | 20 | 4000 | 7.26 | 11.15 |
| 831 | 1131 | 3 | 30 | 6000 | 5.33 | 7.25 |
| 652 | 830 | 4 | 40 | 8000 | 4.18 | 5.32 |
| 533 | 651 | 5 | 50 | 10000 | 3.42 | 4.17 |
| 448 | 532 | 6 | 60 | 12000 | 2.87 | 3.41 |
| 385 | 447 | 7 | 70 | 14000 | 2.47 | 2.87 |
| 335 | 384 | 8 | 80 | 16000 | 2.15 | 2.46 |
| 296 | 334 | 9 | 90 | 18000 | 1.90 | 2.14 |
| 264 | 295 | 10 | 100 | 20000 | 1.69 | 1.89 |
| 237 | 263 | 11 | 110 | 22000 | 1.52 | 1.69 |
| 214 | 236 | 12 | 120 | 24000 | 1.37 | 1.51 |
| 194 | 213 | 13 | 130 | 26000 | 1.24 | 1.37 |
| 178 | 193 | 14 | 140 | 28000 | 1.14 | 1.24 |
| 163 | 177 | 15 | 150 | 30000 | 1.04 | 1.13 |
| 150 | 162 | 16 | 160 | 32000 | 0.96 | 1.04 |
| 139 | 149 | 17 | 170 | 34000 | 0.89 | 0.96 |
| 128 | 138 | 18 | 180 | 36000 | 0.82 | 0.88 |
| 119 | 127 | 19 | 190 | 38000 | 0.76 | 0.81 |
| 110 | 118 | 20 | 200 | 40000 | 0.71 | 0.76 |
| 103 | 109 | 21 | 210 | 42000 | 0.66 | 0.70 |
| 96 | 102 | 22 | 220 | 44000 | 0.62 | 0.65 |
| 89 | 95 | 23 | 230 | 46000 | 0.57 | 0.61 |
| 83 | 88 | 24 | 240 | 48000 | 0.53 | 0.56 |
| 78 | 82 | 25 | 250 | 50000 | 0.50 | 0.53 |
| 73 | 77 | 26 | 260 | 52000 | 0.47 | 0.49 |

| octaden per blok | | blokken per | | | overdrachtsnelheid in msec per blok | |
|---|---|---|---|---|---|---|
| minimum | maximum | spoor | cylinder | module | minimum | maximum |
| 68 | 72 | 27 | 270 | 54000 | 0.44 | 0.46 |
| 64 | 67 | 28 | 280 | 56000 | 0.41 | 0.43 |
| 60 | 63 | 29 | 290 | 58000 | 0.38 | 0.40 |
| 56 | 59 | 30 | 300 | 60000 | 0.36 | 0.38 |
| 53 | 55 | 31 | 310 | 62000 | 0.34 | 0.35 |
| 49 | 52 | 32 | 320 | 64000 | 0.31 | 0.33 |
| 46 | 48 | 33 | 330 | 66000 | 0.29 | 0.31 |
| 43 | 45 | 34 | 340 | 68000 | 0.28 | 0.29 |
| 41 | 42 | 35 | 350 | 70000 | 0.26 | 0.27 |
| 38 | 40 | 36 | 360 | 72000 | 0.24 | 0.26 |
| 36 | 37 | 37 | 370 | 74000 | 0.23 | 0.24 |
| 33 | 35 | 38 | 380 | 76000 | 0.21 | 0.22 |
| 31 | 32 | 39 | 390 | 78000 | 0.20 | 0.21 |
| 28 | 30 | 40 | 400 | 80000 | 0.18 | 0.19 |
| 26 | 27 | 41 | 410 | 82000 | 0.17 | 0.17 |
| 24 | 25 | 42 | 420 | 84000 | 0 15 | 0.16 |
| 22 | 23 | 43 | 430 | 86000 | 0.14 | 0.15 |
| 21 | 21 | 44 | 440 | 88000 | 0.13 | 0.13 |
| 20 | 20 | 45 | 450 | 90000 | 0.13 | 0.13 |
| 18 | 19 | 46 | 460 | 92000 | 0.12 | 0.12 |
| 16 | 17 | 47 | 470 | 94000 | 0.10 | 0.11 |
| 15 | 15 | 48 | 480 | 96000 | 0.10 | 0.10 |
| 13 | 14 | 49 | 490 | 98000 | 0.08 | 0.09 |
| 12 | 12 | 50 | 500 | 100000 | 0.08 | 0.08 |
| 10 | 11 | 51 | 510 | 102000 | 0.06 | 0.07 |
| 9 | 9 | 52 | 520 | 104000 | 0.06 | 0.06 |
| 8 | 8 | 53 | 530 | 106000 | 0.05 | 0.05 |
| 7 | 7 | 54 | 540 | 108000 | 0.04 | 0.04 |
| 5 | 6 | 55 | 550 | 110000 | 0.03 | 0.04 |

Tabel: Capaciteiten en overdrachtsnelheden voor blokken met key (= sleutel).

| octaden per blok | | blokken per | | | overdrachtsnelheid in msec per blok | |
|---|---|---|---|---|---|---|
| minimum | maximum | spoor | cylinder | module | minimum | maximum |
| 1721 | 3605 | 1 | 10 | 2000 | 11.03 | 23.11 |
| 1112 | 1720 | 2 | 20 | 4000 | 7.13 | 11.03 |
| 812 | 1111 | 3 | 30 | 6000 | 5.21 | 7.12 |
| 633 | 811 | 4 | 40 | 8000 | 4.06 | 5.20 |
| 513 | 632 | 5 | 50 | 10000 | 3.29 | 4.05 |
| 429 | 512 | 6 | 60 | 12000 | 2.75 | 3.28 |
| 365 | 428 | 7 | 70 | 14000 | 2.34 | 2.74 |
| 316 | 364 | 8 | 80 | 16000 | 2.03 | 2.33 |
| 276 | 315 | 9 | 90 | 18000 | 1.77 | 2.02 |
| 245 | 275 | 10 | 100 | 20000 | 1.57 | 1.76 |
| 218 | 244 | 11 | 110 | 22000 | 1.40 | 1.56 |
| 195 | 217 | 12 | 120 | 24000 | 1.25 | 1.39 |
| 175 | 194 | 13 | 130 | 26000 | 1.12 | 1.24 |
| 159 | 174 | 14 | 140 | 28000 | 1.02 | 1.12 |
| 144 | 158 | 15 | 150 | 30000 | 0.92 | 1.01 |
| 131 | 143 | 16 | 160 | 32000 | 0.84 | 0.92 |
| 120 | 130 | 17 | 170 | 34000 | 0.77 | 0.83 |
| 109 | 119 | 18 | 180 | 36000 | 0.70 | 0.76 |
| 100 | 108 | 19 | 190 | 38000 | 0.64 | 0.69 |
| 91 | 99 | 20 | 200 | 40000 | 0.58 | 0.63 |
| 83 | 90 | 21 | 210 | 42000 | 0.53 | 0.58 |
| 77 | 82 | 22 | 220 | 44000 | 0.49 | 0.53 |
| 70 | 76 | 23 | 230 | 46000 | 0.45 | 0.49 |
| 64 | 69 | 24 | 240 | 48000 | 0.41 | 0.44 |
| 59 | 63 | 25 | 250 | 50000 | 0.38 | 0.40 |
| 54 | 58 | 26 | 260 | 52000 | 0.35 | 0.37 |

| octaden per blok | | blokken per | | | overdrachtsnelheid in msec per blok | |
|---|---|---|---|---|---|---|
| minimum | maximum | spoor | cylinder | module | minimum | maximum |
| 49 | 53 | 27 | 270 | 54000 | 0.31 | 0.34 |
| 45 | 48 | 28 | 280 | 56000 | 0.29 | 0.31 |
| 41 | 44 | 29 | 290 | 58000 | 0.26 | 0.28 |
| 37 | 40 | 30 | 300 | 60000 | 0.24 | 0.26 |
| 34 | 36 | 31 | 310 | 62000 | 0.22 | 0.23 |
| 30 | 33 | 32 | 320 | 64000 | 0.19 | 0.21 |
| 27 | 29 | 33 | 330 | 66000 | 0.17 | 0.19 |
| 24 | 26 | 34 | 340 | 68000 | 0.15 | 0.17 |
| 21 | 23 | 35 | 350 | 70000 | 0.13 | 0.15 |
| 19 | 20 | 36 | 360 | 72000 | 0.12 | 0.13 |
| 17 | 18 | 37 | 370 | 74000 | 0.11 | 0.12 |
| 14 | 16 | 38 | 380 | 76000 | 0.09 | 0.10 |
| 12 | 13 | 39 | 390 | 78000 | 0.08 | 0.08 |
| 9 | 11 | 40 | 400 | 80000 | 0.06 | 0.07 |
| 7 | 8 | 41 | 410 | 82000 | 0.04 | 0.05 |
| 5 | 6 | 42 | 420 | 84000 | 0.03 | 0.04 |

APPENDIX 2.

Woordenlijst
Deze lijst volgt de termen zoals ze achtereenvolgens in de paragrafen van de syllabus voorkomen; ze dient vooral als hulpmiddel bij het repeteren van de stof.

| Paragraaf: | |
|---|---|
| 1 | Object (entity, begrip, betekenisvolle eenheid) |
| | Kenmerk (property, gegevensklasse) |
| | Logische structuur |
| | Fysieke structuur (opslagstructuur) |
| 2.1 | Logische bouwsteen |
| | Character |
| 2.2 | Item |
| | Naam item (attribuut, property, gegevensklasse! |
| | Waarde item (value, gegevenswaarde) |
| | Veld (rubriek, field) |
| 2.3 | Recordtype (entity) |
| | Sleutelitem (recordsleutel) |
| | Wijzer (pointer, reference) |
| | Blokken |
| | Secundair geheugensysteem |
| | Vastleggen items: Fixed |
| | Indexed |
| | Separator |
| | Label |
| 2.4 | Multifile volume |
| | Multivolume file |
| | Databank |
| | File-activity (relatieve frequentie van bestandsreferenties) |
| | File-volatility |
| | File-turnover (vervangingsgraad) |
| 3.1 | Sequentiële structuur |
| | Boomstructuur |
| | Netwerkstructuur |
| 3.2 | Lijststructuur, sliertstructuur (list) |
| | Enkelvoudige kettingstructuur (ketting in één richting) |
| | Dubbele kettingstructuur (ketting in twee richtingen) |
| | Ring kettingstructuur (circular list) |
| 3.3 | Linear search (sequentieel zoeken) |
| | Full search |
| | Binair zoeken |
| | Skip-sequential (getraps sequentieel) |
| | Directe zoek |
| | Indirecte zoek |

| | |
|---|---|
| 4 | Blok |
| | Blokhiaat, gap (blockspace) |
| | Logical record (logisch record) |
| | Physical record |
| | Grootvader-vader-zoon-systeem |
| | Schijfzijde |
| | Cilinder |
| | Spoor |
| | Batchverwerking (groepsgewijze verwerking) |
| | Schijvengeheugens |
| | Identificatie (key, sleutel) |
| 5.1 | Index-sequentiële organisatie (geïndiceerd sequentiële org.) |
| | Sleutelvolgorde |
| | Cilindertabel (cilinderindex) |
| | Sporentabel (tracktable, spoorindex, trackindex) |
| | Volume-index |
| | Overloopsporen (overflow area, overflow gebied) |
| | Reorganiseren |
| | Normal-entry |
| | Overflow-entry |
| 5.3 | Directe adressering |
| | Indirecte adressering |
| 5.3.1 | Schijfadres |
| | Sleutel |
| | Disk-adres |
| | Relatieve adressering |
| 5.3.2 | Randomizing (sleutelconversie of sleuteltransformatie procedure) |
| | Primair gebied |
| | Bezettingsgraad |
| | Adresseringstechnieken |
| | Cijfer analyse tabel |
| | Extracting |
| | Folding (knippen) |
| | Vermenigvuldiging (midsquaring) |
| | Distribution method |
| | Home-address (huisadres, spoornummer) |
| | Binning |
| | Strook |
| 5.4 | Postgewijs |
| | Groepsgewijs |
| | Deadlock |
| | Resource-sharing |
| | Accesstijd |
| 5.5.1 | Relatieve bestandsorganisatie |
| 5.5.2 | Inverted file structuren (geïnverteerd bestand) |

| | |
|---|---|
| 5.5.3 | Lijststructuren |
| | Keten vrije plaatsen |
| | Boom met meerdere |
| | Niveaus |
| | Produktstructuurbestand |
| | Gerelateerde informatie |
| | Forward chain |
| | Downward chain |
| | Backward chain |
| | Upward chain |
| | Toevoegen in een ketting |
| | Masterrecord |
| | Verwijsadres |
| | Gesegmenteerde records |
| | Master and detail records |
| | Informatie segmenten |
| 5.6 | Sequentiële organisatie |
| | Index-sequentiële organisatie |
| | Direct toegankelijke organisatie (random organisatie) |
| | Blokken |
| | Ontblokken |
| | Parallel buffering (parallel verwerking) |
| | Achtergrondgeheugen |
| | Primair geheugen |
| | Sleutelwaarden |
| | Bestandsbeveiliging |
| | Grootvader, vader, zoon-systeem |
| | Dump |
| | At random verwerken |
| 6 | Bestandsontwerp |
| | Bestandsdynamiek |
| 6.1.2 | Splitting files |
| | Processing cycle files |
| | Prime files |
| | Trailer files |
| | Split files |
| 6.1.3 | File turner |
| 6.1.4 | File activity |
| 6.1.5 | File volatility |
| 6.2 | Hoofdbestand (master file, permanent bestand) |
| | Gerelateerd bestand (related file) |
| | Geïnverteerd bestand (inverted file) |
| | Transactiebestand (transaction file) |
| | Inputbestand |
| | Mutatiebestand |
| | Inquiry record |
| | Activity record |

Outputbestand
Historisch bestand
Totaalbestand
Summary file
Recovery bestand
Backup bestand
Data base
Primair bestand
Trailer bestand
Data Base Handler
Data Language
Partitioned bestand
Members
Catalogus (directory)
6.3 File consolidation
Bestandsnazorg
File maintenance
6.4.1 Veldontwerp (field design)
Record compaction
Sector
Spoor
Decimal scaling
Floating point
Variabele lengte velden
Bitting
Coding
Heading
Substituering
Collating sequence
6.4.2 Minor sorteerveld
Major sorteerveld
6.4.3 Limiet van het bestand
Data management software
Fysisch record
Interrecord gaps
Accesstijd
Identificatie records (header label)
7 File control
Controle van het bestand
7.1 Interne controle
Positieve controlemiddelen
Negatieve controlemiddelen
7.2 Ingebouwde controles
Machine gerichte controle
7.2.1 Hardware controles
Hole count
Protectiëring

| | |
|---|---|
| | Pariteitscontrole |
| | Overflow |
| | Geheugenprotectie |
| 7.2.2 | House keeping |
| | Header-label |
| | Bewaartermijn |
| | Haspelnummer |
| | Volledigheid controle |
| | Block totals |
| | Record totals |
| | Hash totals |
| | Trailer label |
| | Utility programs |
| | Sluitkaart |
| | Sluitblok |
| | Checkpoint |
| | Restart |
| | Read after write |
| | Read errors |
| | Wrong length record |
| 7.3 | Geprogrammeerde controles |
| 7.3.1 | Invoercontroles |
| | Plaatscontrole |
| | Limietcontrole |
| | Volgordecontrole |
| | Geldigheidscontrole |
| | Afhankelijkheidscontrole |
| | Volledigheidscontrole |
| | Self-checking codes |
| 7.3.2 | Vierkantscontrole |
| | Audit trail |
| | Cross footing |
| | Leap-frog |
| 8.1.1 | Database |
| 8.1.3 | Database systeem |
| | Database management systemen (DBMS) |
| 8.1.4 | Codasyl DBTG |
| 8.1.5 | Gastheertaal (host language) |
| | Gastheertaal systeem |
| | Selfcontained systeem |
| | Data Definition Language (DDL) |
| | Device Media Control Language (DMCL) |
| | Data Manipulation Language (DML) |
| 8.2.1 | Entiteit |
| | Entiteittype |
| | Verschijningsvorm (occurrence) |
| | Entiteitenmodel |

Eigenschap (attribute, property)
Eigenschaptype
Eigenschapswaarde
Informatie
Relatie
8.2.2 Data-item
Data-aggregate
Record
Vector
Repeterende groep
8.2.3 Set
Owner
Member
Set verschijningsvorm
Bachman-diagram
8.2.4 Gegevensstructuur (data structure)
Schema
Sequentiële structuur
Boomstructuur
Netwerkstructuur
Automatic membership
Manual membership
Mandatory membership
Optional membership
8.2.5 Area
Exclusive access
Protected access
8.2.6 Search key
8.3.1 Fysieke opslag
8.3.2 Database key
Chaining
Pointer array
Fysiek sequentieel
8.3.3 Storage Structure Language (SSL)
8.4 Betrouwbaarheid (reliability)
Availibility
8.4.1 Geheimhouding (privacy)
8.4.2 Veiligheid (security)
8.4.3 Accuratesse (accuracy)
8.4.4 Integriteit (integrity)
8.4.5 Consistentie (consistency)
8.5.1 Subschema
Subschema DDL
8.5.2 Sequentiële accessvorm
Directe accessvorm

| | |
|---|---|
| 8.5.3 | Gemeenschappelijk gebruik (concurrent use) |
| | Transactie |
| | Integriteitseenheid |
| | Deadlock |
| | Exclusieve toegang (exclusive access) |
| | Beschermde toegang (protected access) |
| 8.5.4 | DML |
| | Open |
| | Close |
| | Find |
| | Modify |
| | Store |
| | Delete |
| | Insert |
| | Remove |
| | Move |
| 8.5.5 | Database Handler (DBH) |
| | User Working Area (UWA) |
| | Message handler |
| | DB/DC processing |
| 8.6.4 | Check pointing |
| | Logging |
| | Back-up |
| | Before image |
| | After image |

# APPENDIX 3.

## Opgaven

1. Gegeven een tape, 800 bytes/inch, 2400 feet lang, lengte van interrecord gap: 0.6 inch.
   Stel we hebben een bestand met logische records van vaste lengte (200 karakters per record), terwijl de blokkingsfactor 5 is. Bereken hoeveel logische records deze tape bevatten kan, als men weet dat 30 feet gereserveerd is voor header- en trailerlabel.

2. Gegeven een disk-pack, met 200 cilinders, 20 tracks per cilinder, 7294 bytes per track, rotatiesnelheid 25 msec.
   Gegeven een bestand, bestaande uit 10.000 records, 200 bytes per record.

   a. Als we 23 records/track wensen, hoeveel cilinders beslaat dit bestand dan?

   b. Stel dat het bestand sequentieel ligt opgeslagen vanaf cilinder 0, track 0, hoeveel accessbewegingen zijn dan nodig om de gehele file sequentieel af te werken?

   c. Het antwoord op vraag b toont aan dat accesstijd verwaarloosbaar is bij sequentiële afwerking. De meeste tijd kost dan de rotatie en de data-overdracht.
      Als we voor elk record een volledige omwenteling rekenen, hoeveel kost dan het lezen van de gehele file?

De volgende opgaven betreffen een bestand met 10.000 logische records; elk record bevat 160 karakters (lengte = 160 bytes of octaden), daarbij inbegrepen een 7-byte sleutel.
Het opslagmedium is een schijvengeheugen met de volgende eigenschappen (IBM 2311):

| | |
|---|---|
| aantal cilinders | : 200 |
| aantal sporen per cilinder | : 10 |
| aantal bytes (char) per spoor | : 3625 |
| gemiddelde accesstijd | : 75 msec. |
| duur van een rotatie | : 25 msec. |
| overdrachtssnelheid | : 156 kbytes |

Zie ook de tabel op de volgende bladzijde.
De kolom met (K) betekent: 'formatted with key'; de andere kolom betekent: 'formatted without key'.
Records worden veelal op twee manieren op een spoor gerepresenteerd:
- naast administratieve informatie over plaats en lengte van het record worden de gegevens fysisch als een geheel opgeslagen (formatted without key);

| Max. bytes per record | Records per track | Max. bytes per record (K) |
|---|---|---|
| 3625 | 1 | 3605 |
| 1740 | 2 | 1720 |
| 1131 | 3 | 1111 |
| 830 | 4 | 811 |
| 651 | 5 | 632 |
| 532 | 6 | 512 |
| 447 | 7 | 428 |
| 384 | 8 | 364 |
| 334 | 9 | 315 |
| 295 | 10 | 275 |
| 263 | 11 | 244 |
| 236 | 12 | 217 |
| 213 | 13 | 194 |
| 193 | 14 | 174 |
| 177 | 15 | 158 |

Tabel 1: Tabel voor spoorcapaciteit.

- aan de administratieve informatie wordt de sleutel van het record toegevoegd; de gegevens van het record worden weer als een geheel opgeslagen (formatted with key).

Voor fysische records met een lengte kleiner dan aangegeven in de tabel hierboven kan men gebruik maken van de volgende formules:

$$B = \frac{\text{benodigde hoeveelheid}}{\text{bytes voor een record}} = 61 + \frac{537 * DL}{512}$$ (formatted without key) (1)

$$B = \frac{\text{benodigde hoeveelheid}}{\text{bytes voor een record}} = 81 + \frac{537 * (DL+KL)}{512}$$ (formatted with key) (2)

$$C = \frac{\text{benodigde hoeveelheid}}{\text{bytes voor het \underline{laatste} record}} = DL$$ (formatted without key) (3)

$$C = \frac{\text{benodigde hoeveelheid}}{\text{bytes voor het \underline{laatste} record}} = 20 + KL + DL$$ (formatted with key) (4)

De getallen geven de benodigde ruimte voor gaps en administratie weer.
DL = data length = aantal bytes van het record.
KL = key length = aantal bytes van de sleutel.

3. Het bovengenoemde bestand is sequentieel georganiseerd.

   a. Wat is de optimale blokkingsfactor als we een blok niet groter dan 1000 bytes willen maken?

b. Hoeveel cilinders beslaat de file bij deze blokkingsfactor?

c. Hoeveel tijd kost het sequentieel lezen van de file? (Ga er van uit dat elke leesopdracht een omwenteling kost; dit is aan de hoge kant.)

d. Hoeveel tijd kost het random lezen van 5000 records?

4. Het bovengenoemde bestand is nu index-sequentieel georganiseerd; blokkingsfactor is weer 5; bestand krijgt een aaneengesloten stuk geheugen.
De bijgeleverde standaard software heeft de volgende eigenschappen:
- per cilinder is een track gereserveerd voor overflow;
- bij elke cilinder (is) zijn de eerste track(s) gereserveerd voor de track-index;
- elke track-index bevat als eerste record een set gegevens die de overflow controleert (COCR); vervolgens bevat de track-index paren entries (een voor primaire spoor, een voor overloop vanuit dit spoor).
  Elke entry geeft weer: adres van de track, hoogste sleutelwaarde op die track.
  Adres van de track: 10 bytes
  Sleutelwaarde : door gebruiker op te geven;
- elke track-index wordt afgesloten door een dummy-entry (gevuld met 'nep'-adres en 'nep'-sleutel) als eindsymbool;
- cilinder-index heeft dezelfde opbouw als de track-index;
- bij geblokte records wordt de hoogste sleutel van de logische records in het blok als sleutel van het blok beschouwd en hieraan toegevoegd;
- overflow records zijn niet geblokt;
- aan een overflow record wordt verwijsadres (10 bytes) toegevoegd.

a. Hoeveel logical records passen op een primair spoor?

b. Hoeveel logical records passen op een overflow spoor?

c. Hoeveel primair sporen bevat een cilinder?

d. Hoeveel entries zal de track-index bevatten, afgezien de COCR?

e. Hoeveel bytes zijn nodig voor de index-entries voor elke track-index (gebruik de hierboven gegeven formules)?

f. Hoeveel logical records passen nog op de tracks, die de track-index bevatten?

g. Afgezien van cilinder-index en overflow-area, hoeveel cilinders heeft deze file nodig?

h. Hoeveel entries zal de cilinder-index bevatten?

i. Hoeveel sporen heeft de cilinder-index nodig?

j. Hoeveel disk-tijd kost het sequentieel lezen van de file (neem aan dat de overflow-area voor de helft gevuld is)?

k. Hoeveel disk-tijd is nodig om 5000 records random te lezen? (Cilinder-index is niet in het kerngeheugen, maar wordt met hetzelfde accessmechanisme gelezen als de rest van de file.)

l. Hoeveel disk-tijd is nodig om 5000 records te lezen als de cilinder-index door een ander accessmechanisme gelezen kan worden als de rest van de file?

m. Hoeveel disk-tijd is nodig om 5000 records te lezen als de cilinder-index in het kerngeheugen is?

5. Het bestand is nu 'direct toegankelijk' georganiseerd.
De records zijn niet geblokt en 'formatted with key' opgeslagen.
De bezettingsgraad van het primaire gebied mag niet hoger dan 85% zijn.
De adressering is indirect. De overflow is niet gebaseerd op synoniemketens, maar wordt opgevangen door synoniemen zo dicht mogelijk bij het berekende adres te plaatsen.

   a. Hoeveel cilinders heeft deze file nodig?

   b. Hoeveel disk-tijd is nodig om de file sequentieel te verwerken? (Neem aan dat een adrestabel gebruikt is en dat het 1 zoekopdracht en 1.2 leesopdrachten kost om elk record te vinden.)

   c. Hoeveel disk-tijd is nodig om 5000 records random te lezen?

6. Vergelijk door opgave 3, 4 en 5 te beschouwen de drie bestandsorganisatietechnieken ten aanzien van geheugenbeslag en verwerkingssnelheid bij sequentiële en directe verwerking.

Antwoorden bij opgaven

1. Dichtheid : 800 bytes/inch—0.00125 inch/kar.
Tapelengte : 2400 * 12 = 28800 inch.
Voor informatie beschikbaar: 28800 - (12 * 30) = 28440 inch.
Per blok nodig: gaplengte + aantal kar/blok * inch/kar = 0.6 + 0.00125 * N. N = 5 * 200 = 1000 kar.

Tape kan $\frac{28440}{0.6+0.00125*1000}$ = 15.373 blokken bevatten.

Dit is 5 * 15.373 = 76.865 logische records.

Formule:

$$\text{Aantal blokken} = \frac{\text{tapelengte}}{\text{lengte gap + lengte kar. * aantal kar/blok}}$$

2. a. 10.000/23 = 435 tracks,
435/20 = 22 cilinders.

b. 21 bewegingen * 25 msec. = 525 msec.

c. (10.000 * 25) = 250.000 msec. = $\frac{250.000}{60.000}$ = 4.2 min.

3. a. We kunnen kiezen uit blokkingsfactoren 1, 2, 3, 4, 5, 6, die alle bloklengtes < 1000 bytes opleveren.
De tabel op pagina 149 leert dat de blokkingsfactoren 4 en 5 (640 en 800 bytes per fysisch record) beide 20 logische records per spoor opleveren. (Immers bij 640 bytes kan een spoor 5 en bij 800 bytes 4 fysische records bevatten.)
We kiezen 5, omdat een hogere blokkingsfactor gunstiger is (buffergebruik, vermindering gaps, etc.).

b. 10.000/20 = 500 sporen = 500/10 = 50 cilinders.

c. Zoeken: (geschiedt sequentieel—min. accesstijd)
50 * 25 msec. = 1250
Lezen: 10.000/5 = 2000 blokken
2000 * 25 msec. = 50.000
51.250 msec. = 51.250/60.000 = 0.9 min.

d. Gemiddeld genomen zal voor het zoeken van elk record het halve bestand gelezen moeten worden.
Per record is dit: 51.250/2 = 25.625 msec./record.
Totaal: 5.000 * 25.625 = 128.125 sec. = 35.6 uur.

4. a. Bloklengte = 5 * 160 + 7 (sleutellengte; hoogste sleutel van records uit blok toevoegen!).
= 807.
Tabel 1 leert: 4 van deze fysische records per spoor.
Dus: 20 logische records.

b. Overflow records zijn ongeblokt.
Bloklengte = 160 + 7 (sleutel) + 10 (verwijsadres) = 177.
Tabel 1 leert: 13 van deze records per track (kijk bij *formatted with key!*).
*Dus: 13 logische records.*

c. 10 sporen per cilinder - 1 spoor voor track-index - 1 spoor voor overflow = 8.

d. 2 entries voor records op het stuk dat overblijft achter de track-index.
8 * 2 voor de records op de primaire sporen.
1 dummy entry (afsluiten track-index)
19 entries.

e. Gebruik formule 2 (formatted with key).
Track-index entries zijn in feite ook records:
sleutel = hoogste sleutel op dat spoor
data = adres van het spoor.
Index-entry kost $81 + \frac{537(17)}{512} = 98$ bytes (DL = 10, K = 7).
Totaal 19 * 98 = 1862 bytes.

f. Spoorcapaciteit: 3625.
Over 3625 - 1862 (index grootte) = 1763 bytes.
1 blok = 5 * 160 + 7 = 807 bytes. Er kunnen er dus hooguit 2 in.
Nagaan:
Formule (2) met DL = 800 en KL = 7: $81 + \frac{537*(800+7)}{512} = 927$ bytes.
Formule (4) (laatste record) : 20 + 7 + 800 = 827 bytes.
927 + 827 = 1754 bytes. Kan er in.
*Dus:* 10 logische records passen nog in spoor van de track-index.

g. (20 * 8) + 10 = 170 logische records per cilinder.
10.000/170 = 59 cilinders.

h. 60 entries: 1 voor elke cilinder + dummy entry.

i. Entry bestaat uit sleutel (KL = 7) en adrescilinder (DL = 10) met formules (2) en (4):
index-entry $= 81 + \frac{537(17)}{512} = 98$
laatste entry = 20 + 7 + 10 = 37.
Een track bevat dan:
$$\text{laatste record} + \frac{\text{cap. track - aantal bytes voor laatste record}}{\text{aantal bytes voor een record}}$$
In dit geval: $1 + \frac{3625 - 37}{98} = 37$.
Een track bevat 37 entries.
De cilinder-index (60 entries) past dus op 2 sporen.

j. Overflow ruimte voor deze file bevat 59 sporen (1 per cilinder), per spoor 13 records—59 * 13 = 767.
De helft zou bezet zijn: ± 400 records.
Gaan we vervolgens niet uit van 9.600 records, maar 10.000, dan levert dit 2.000 fysische records op.
Nemen we weer aan dat lezen een volle rotatie kost:
- 2000 * 25 + 400 * 25 = 60.000 msec.
- zoeken en lezen van een paar entries uit de track-index:
  10 * 59 * 12.5 = 7375 msec.
- zoeken van de volgende cilinder: 59 * 25 = 1475 msec.
Totaal: 68850 msec. = 1.15 minuten.

k. Zoeken van cilinder-index : 75 msec. (gem. accesstijd)
Zoeken van de cilinder : 75 msec.
Lezen van de track-index entry: 12.5 msec.
Lezen van cilinder-index entry: 12.5 msec.
Lezen van record : 25 msec.
200 msec.
5.000 * 200/60.000 = 16.6 minuten.

l. Dit scheelt bij het zoeken van de cilinder-index; je kunt access daarop instellen:
zoeken van de cilinder : 75 msec.
lezen track-index entry : 12.5 msec.
lezen cilinder-index entry : 12.5 msec.
lezen van record : 25 msec.
125 msec.
125 msec. * 5.000/60.000 = 10.4 minuten.

m. Dit schakelt ook nog het lezen van de cilinder-index entries uit:
zoeken cilinder : 75 msec.
lezen track-index entry : 12.5 msec.
lezen record : 25 msec.
112.5 msec.
112.5 * 5.000/60.000 = 9.38 minuten.

5. a. Bezettingsgraad: 85%— ± 11765 locaties voor records zijn nodig.
Per track kunnen we 14 records kwijt (immers 1 record: 160 + 7 = 167 bytes; een track kan er dan 14 bevatten (zie tabel 1)).
Dit levert: 11.765/14 —841 tracks ; 841/10 —85 cilinders.

b. Per record: zoek : 75 msec. +
lezen: 1.2 * 25 = 30 msec. = 105 msec.
Voor de gehele file: 105 * 10.000/60.000 = 17.5 minuten.

c. Per record : 105 msec.
Voor 5.000 records: 105 * 5.000/60.000 = 8.75 minuten.

| 6. | SEQ. | IND. SEQ | RANDOM |
|---|---|---|---|
| benodigde aantal cilinders | 50 | 59 | 85 |
| benodigde tijd voor sequentiële verwerking van de gehele file (minuten) | 0.9 | 1.15 | 17.5 |
| benodigde tijd voor random-verwerking van 5.000 records (minuten) | - | 9.38+16.6* | 8.75 |

*Afhankelijk van plaats van cilinder-index.

LITERATUUR


| | |
|---|---|
| - ARDI Systems Handbook | (Philips Electrologica). |
| - Data File Handbook | (IBM, C20-1638). |
| - Bestandsorganisatie | Lunbeck, R.J. en Remmen, F., Den Haag, Academic Service, 1975 |
| - DBTG-proposal | (Codasyl Data Base Task Group) |
| - Introduction to Pholas | (Philips Electrologica). |

## ACADEMIC SERVICE INFORMATICA UITGAVEN

### AUTOMATISERING EN COMPUTERS

*Computers en onze samenleving* - M.A. Arbib
*Computers in de negentiger jaren* - G.L. Simons
*De informatiemaatschappij* - Jan Everink
*Basiskennis informatieverwerking* - Jan Everink
*AIV, Automatisering van de informatieverzorging* - Th.J.G. Derksen en H.W. Crins
*Organisatie, informatie en computers* - D.M. Kroenke
*De Viewdata revolutie* - S. Fedida en R. Malik

### MICROCOMPUTERS

*Microcomputers thuis en op school* - K.P. Goldberg en R.D. Sherwood
*Bouw zelf een Expertsysteem in BASIC* - C. Naylor
*Programmeercursus Microsoft BASIC* - Nok van Veen
*Werken met bestanden in BASIC* - L. Finkel en J.R. Brown
*40 Grafische programma's voor de Commodore 64* - M. Sutter
*Doe-het-zelf programma's op de Commodore 64* - D. Kreutner
*Programmeercursus BASIC op de Commodore 64* - Nok van Veen
*TRS-80 BASIC* - Bob Albrecht e.a.
*TRS-80 BASIC voor gevorderden* - Don Inman e.a.
*Exidy sorcerer en BASIC* - Nok van Veen e.a.
*40 Grafische programma's voor de Electron en BBC* - M. Sutter
*Het Electron en BBC Micro boek* - Jim McGregor en Alan Watt
*Ontdek de ZX-Spectrum* - Tim Hartnell
*Werken met bestanden op de Apple* - L. Finkel en J.R. Brown
*Programmeercursus Applesoft BASIC* - Nok van Veen
*40 Grafische programma's voor de Apple II,IIe,IIc* - M. Sutter
*40 Grafische programma's in MSX BASIC* - M. Sutter
*Programmeercursus MSX BASIC* - Nok van Veen

### MICROPROCESSORS EN ASSEMBLEERTALEN

*Procescomputers, basisbegrippen* - dr.ir. J.E. Rooda en ir. W.C. Boot
*Cursus Z-80 assembleertaal* - Roger Hutty
*6502 Assembleertaal en machinecode voor beginners* - A.P. Stephenson

### BESTURINGSSYSTEMEN

*Inleiding besturingssystemen* - A.M. Lister
*Systeemprogrammatuur en software-ontwikkeling voor microcomputers* - E. Verhulst
*Bedrijfssystemen* - EIT-serie, deel 4
*CP/M het operating system voor microcomputers* - J.N. Fernandez en R. Ashley
*CP/M 86* - Nok van Veen
*CP/M voor gevorderden* - A. Clarke e.a.
*PC DOS, het besturingssysteem van de IBM PC* - R. Ashley en J.N. Fernandez
*MS/DOS, het besturingssysteem voor 16 bit microcomputers* - R. Ashley en J.N. Fernandez
*UNIX, het standaard operating system* - G.J.M. Austen en H.J. Thomassen
*Werken met UNIX* - Brian W. Kernighan en Rob Pike

### PERSONAL COMPUTERS

*Het werken met bestanden op de IBM PC* - L. Finkel en J.R. Brown
*De IBM PC en zijn toepassingen* - Laurence Press
*40 Grafische programma's voor de IBM PC* - M. Sutter
*Werken met VisiCalc* - C. Klitzner en M.J. Plociak
*Multiplan, een hulpmiddel bij de bedrijfsvoering* - D.F. Cobb e.a.
*Multiplan diskettes*
*Werken met Lotus 1-2-3* - D. Cobb en G. LeBlond

### PROGRAMMEREN

*Een methode van programmeren* - prof.dr. Edsger W. Dijkstra en ir. W.H.J. Feijen
*Programmeren, het ontwerpen van algoritmen (met Pascal)* - ir. J.J. van Amstel
*Inleiding tot het programmeren, deel 1* - ir. J.J. van Amstel
*Inleiding tot het programmeren, deel 2* - ir. J.J. van Amstel
*Programmeren, deel 2: van analyse tot algoritme* - prof.drs. C. Bron
*Inleiding programmeren en programmeertechnieken* - EIT-serie, deel 1
*Het Groot Pascal Spreuken Boek* - H.F. Ledgard e.a.
*JSP - Jackson Struktureel Programmeren* - Henk Jansen
*JSP Uitwerkingenboek* - Henk Jansen

## PROGRAMMEERTALEN

*Aspecten van programmeertalen* - ir. J.J. van Amstel en ir. J.A.A.M. Poirters
*Programmeertalen, een inleiding* - ir. J.J. van Amstel e.a.
*BASIC* - EIT-serie, deel 3
*Cursus BASIC, een practicum-handleiding voor BASIC op de PRIME* - ir. R. Bloothoofd e.a.
*Cursus Pascal* - prof.dr. A. van der Sluis en drs. C.A.C. Görts
*Cursus eenvoudig Pascal* - prof.dr. A. van der Sluis en drs. C.A.C. Görts
*Inleiding programmeren in Pascal* - C. van de Wijgaart
*Systeemontwikkeling met Ada* - Grady Booch
*Cursus COBOL* - A. Parkin
*Cursus FORTRAN 77* - J.N.P. Hume en R.C. Holt
*Aanvulling cursus FORTRAN 77 voor PRIME-computers* - ing. J.M. den Haan
*De programmeertaal C* - ir. L. Ammeraal
*Flitsend Forth* - Alan Winfield
*Programmeren in LISP* - prof.dr. L.L. Steels

## GEGEVENSSTRUCTUREN EN BESTANDSORGANISATIE

*Informatiestructuren, bestandsorganisatie en bestandsontwerp* - EIT-serie, deel 5
*Programmeren, het ontwerpen van datastructuren en algoritmen* - ir. J.J. van Amstel e.a.
*Bestandsorganisatie* - prof.dr. R.J. Lunbeck en drs. F. Remmen

## DATABASE EN GEGEVENSANALYSE

*Database, een inleiding* - C.J. Date
*Databases* - drs. F. Remmen
*Gegevensanalyse* - R.P. Langerhorst

## INFORMATIE-ANALYSE EN SYSTEEMONTWERP

*Effectieve toepassingen van computers* - M. Peltu
*Voorbereiding van computertoepassingen* - prof.dr. A.B. Frielink
*Systeemontwikkeling volgens SDM* - H.B. Eilers
*Samenvatting SDM* - Pandata
*Informatie-analyse volgens NIAM* - J.J.V.R. Wintraecken
*Evaluation of methods and techniques for the analysis, design and implementation of information systems* - ed. J. Blank en M.J. Krijger
*Inleiding systeemanalyse, systeemontwerp* - W.S. Davis
*Systeemontwikkeling Zonder Zorgen* - Paul T. Ward
*Het ontwerpen van interactieve toepassingen en computernetwerken* - J.A. Scheltens
*EDP Audit* - prof.dr. C. de Backer
*Prototyping, een instrument voor systeemontwerpers* - ed. T. Hoenderkamp en H.G. Sol
*Simulatie, een moderne methode van onderzoek* - drs. S.K. Boersma en ir. T. Hoenderkamp

## EXPERT SYSTEMEN EN KUNSTMATIGE INTELLIGENTIE

*Computerschaak* - H.J. van den Herik
*Expert systemen* - Henk de Swaan Arons en Peter van Lith

## THEORETISCHE INFORMATICA EN SYSTEEMPROGRAMMATUUR

*Informatica, een theoretische inleiding* - dr. L.P.J. Groenewegen en prof.dr. A. Ollongren
*Systeemprogrammatuur* - drs. H. Alblas
*Vertalerbouw* - H. Alblas e.a.

## AANVERWANTE ONDERWERPEN EN OVERIGE TITELS

*Lineaire programmering als hulpmiddel bij de besluitvorming* - prof.dr. S.W. Douma
*Inleiding programmeren* - prof.dr. R.J. Lunbeck
*Analyse van informatiebehoeften en de inhoudsbeschrijving van een databank* - prof.dr. P.G. Bosch en ir. H.M. Heemskerk
*Gegevensstructuren* - R. Engmann e.a.
*Cases en Uitwerkingenboek bij Cases* - prof.dr. P.G. Bosch en H.A. te Rijdt
*De tekstmachine* - dr. M. Boot en drs. H. Koppelaar
*Abstracte automaten en grammatica's* - prof.dr. A. Ollongren en ir. Th.P. van der Weide
*Onderneming en overheid in systeem-dynamisch perspectief* - red. A.F.G. Hanken e.a.
*Simulatie en sociale systemen* - red. J.L.A. Geurts en J.H.L. Oud
*Struktuur en stijl in COBOL* - ir. E. Dürr en dr.ir. F. Mulder
*Cursus ALGOL 60* - prof.dr. A. van der Sluis en drs. C.A.C. Görts